HIGHWAY INFORMATION SYSTEM

RELEASE 4.Ø

RECORD FORMATS AND
SUBROUTINES

Department of Civil Engineering & Engineering Mechanics Montana State University Bozeman

HIGHWAY INFORMATION SYSTEM

RELEASE 4.Ø

RECORD FORMATS AND
SUBROUTINES

Prepared for the:

STATE OF MONTANA
DEPARTMENT OF HIGHWAYS
PLANNING AND RESEARCH BUREAU

In cooperation with the:

U.S. DEPARTMENT OF TRANSPORTATION
FEDERAL HIGHWAY ADMINISTRATION

The opinions, findings, and conclusions expressed in this publication are those of the authors and not necessarily those of the Montana Department of Highways or the Federal Highway Administration.

Project Director: Ralph W. Zimmer, P.E.

Prepared by:

Larry J. Coats, Edward G. Knoyle, and Ralph W. Zimmer

DEPARTMENT OF CIVIL ENGINEERING AND ENGINEERING MECHANICS
MONTANA STATE UNIVERSITY
Bozeman, Montana 59715

April, 1976

FOREWORD

This report is a portion of the documentation of Release 4.Ø of the Highway Information System undertaken by the Department of Civil Engineering and Engineering Mechanics, Montana State University. The retrieval system has been evolving over the last several years under the sponsorship of the Planning and Research Bureau of the Montana Department of Highways with some assistance from the Highway Traffic Safety Division, Montana Department of Community Affairs.

Release 4.Ø of the Highway Information System is documented in the following volumes:

<u>Highway Information System Release 4.Ø: System Overview</u>

Provides an introduction to the Highway Information System.

<u>Highway Information System Release 4.Ø: Index</u>

Provides an index to all manuals except the System Overview and Program Listings.

<u>Highway Information System Release 4.Ø: User's Manual</u>

Describes how to use the Highway Information System for retrieving information and for printing reports and summaries.

<u>Highway Information System Release 4.Ø: Data Coding Manual</u>

Describes the data card formats for entering data into the Highway Information System files.

<u>Highway Information System Release 4.Ø: System Maintenance Manual</u>

Provides information for performing scheduled system backups and file reorganizations and for allocating system files.

<u>Highway Information System Release 4.Ø: Record Formats & Subroutines</u>

Describes the internal record formats of the various files and provides calling sequences to subroutines. This manual is intended for persons writing new programs to add to the Highway Information System.

<u>Highway Information System Release 4.Ø: Programming Details</u>

Describes the existing programs and provides a guide to the program listings. This manual is intended for persons maintaining existing software in the Highway Information System.

<u>Highway Information System Release 4.Ø: Program Listings</u>

Contains computer-generated listings of all source programs of the Highway Information System.

Although the project was conceived, initiated, and primarily funded through the Planning and Research Bureau of the Montana Department of Highways, the development cost of selected portions of the system was borne by the Highway Traffic Safety Division of the Montana Department of Community Affairs.

In developing the system, the CE & EM Department has had the privilege of using an IBM OS/VS1 370/145 computer located at the Data Processing Bureau of the Montana Department of Highways in Helena. PL/I has been used for most of the programs because of its versatility and ease of use. BAL (assembler) has been used for most input-output modules and for other modules that require its increased capabilities and efficiency over PL/I.

The project could never have progressed to its current state without the continued and patient encouragement and assistance from the Planning and Research Bureau and the Data Processing Bureau of the Montana Department of Highways, and from the Highway Traffic Safety Division of the Department of Community Affairs.

The project conclusion was also hastened by the significant effort of other project personnel: Scott H. Danforth, R. Helene Knowlton, and Doug M. Geiger.

TABLE OF CONTENTS

LIST OF CHAPTER HEADINGS

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

This manual is intended for use by persons writing programs for incorporation into the Highway Information System. It contains information on how to compile and link-edit programs, record formats of the Highway Information System files, and subroutines available for use in the system.

This manual is also intended for use by persons modifying existing programs in the system. The two additional manuals are also needed: Highway Information System - Programming Details and Highway Information System - Program Listings.

The System Libraries

The Highway Information System programs are kept in several libraries. These libraries are:

PANVALET library - source modules
HIS.OBJECT library - object modules (temporary storage)
HIS.REL4PTØ library - load modules
HIS.LOADTST library - load modules (for testing)
HIS.SUBRTN4 library - automatic-call library

The PANVALET Library - All of the system's source modules are kept in the PANVALET system belonging to the Montana Department of Highways. Modules that are in production status are kept in an inactive tape file. Modules that are being worked on are kept in an active disk file. Production modules are also kept in a separate backup system maintained by Highway Information System personnel.

The HIS.OBJECT Library - Object modules are intermediate modules produced by compilers. The object modules are processed by the linkage editor to produce load modules. After a load module has been produced and tested, the object module has no further use. Object modules of the Highway Information system are not retained after the load modules are stored in the permanent load module library.

The HIS.REL4PTØ Library - This library is the permanent load module library. Only load modules that have been tested and given production status are stored in this library.

The HIS.LOADTST Library - This library is the temporary load module library. A program in development status is link-edited to this library for testing purposes. When the program is fully tested, it is copied to the HIS.REL4PTØ library and then deleted from HIS.LOADTST.

The HIS.SUBRTN4 Library - This library is the automatic-call library. Subroutines can be stored in load module format in the library to allow future inclusion by automatic call when link-editing.

Compilation of a Source Module

The following steps are required in order to compile a source module:

1. PANVALET step - Generates a scratch file containing the source module being compiled.
2. Compilation step - Reads the source module and produces a scratch file containing an object module.
3. Object step - Stores the object module in the HIS.OBJECT library.
4. (If submitted from a CRJE terminal) Error message step - Writes any error messages to the terminal.

The PANVALET step consists of executing the PAN#1 program of the PANVALET system. An input stream consisting of one or more PANVALET commands must be entered. The PANVALET commands are described in the publication PANVALET User Reference Manual published by Pansophic Systems, Incorporated. At least one ++WRITE WORK or ++INSERT WORK command must be included in the PANVALET step.

The compilation step consists of executing the appropriate compiler. The input to this step is the scratch file built by the PANVALET step. The output object module is written to a scratch file. The object module is not written directly to the HIS.OBJECT library because this would necessitate the parameter DISP=OLD on the HIS.OBJECT DD statement, and would prevent two separate jobs that compile HIS programs to run at the same time.

The object step stores the object module in the HIS.OBJECT library. The copy is done in such a manner that two or more jobs can perform compilations at the same time without the possibility of losing any object modules.

If a compilation is submitted from a CRJE terminal, an error message step can be included as part of the compilation. During this step, any error messages produced by the compiler are written to the terminal. The complete source listings (including error messages) are still written to the line printer.

The HIS.OBJECT library is periodically compressed by system maintenance personnel. During periods of heavy use, however, the library may become full. If this occurs, all compilations will fail with a system D37 abend during the object step. Notify the personnel in charge of library maintenance, and re-submit any compilations that failed after the situation is corrected.

PL/I Compilation - The two procedures HIS4VO and HISCVO are used to compile a PL/I source module. Use HISCVO only when submitting a compilation from a CRJE terminal and if you want compilation error messages to be written to your terminal. Use of the two procedures is identical. Submit the following job setup:

```
// JOB
// EXEC HIS4VO,DATASET=object-module-name
//SYSIN DD *
    PANVALET commands
/*
```

The name specified in DATASET is an 8-character or shorter name that becomes the member name of the object module in the HIS.OBJECT library. The PANVALET commands must include at least one ++WRITE WORK or ++INSERT WORK command. The following optional parameters can be specified on the EXEC statement:

OLIB='library' - Specifies a library other than HIS.OBJECT

C=CT - Enables use of execution-time COUNT option (this option will result in an inefficient module that should be used only for test purposes).

F=FLOW - Enables use of execution-time FLOW option (this option will result in an inefficient module that should be used only for test purposes).

G=GS - Enables the GOSTMT option that allows execution-time error messages to print statement numbers (this option will result in an inefficient module that should be used only for test purposes).

L=NOLIST - Suppresses object listing.

M=NOMAP - Suppresses static storage map listing.

O=NOBJ - Suppresses storage of an object module in HIS.OBJECT.

OPT='OPT(2)' - Requests optimization for fast execution time instead of for small core requirement.

FORM='(class,,form)' - Requests special form in line printer.

COPIES=n - Requests n copies of the source listing (HIS4VO procedure only).

BAL Compilation - The two procedures HIS4VA and HISCVA are used to compile assembler source modules. Use HISCVA only when submitting a compilation from a CRJE terminal and if you want compilation error messages to be written to your terminal. Use of the two procedures is identical. Submit the following job setup:

```
// JOB
// EXEC HIS4VA,DATASET=object-module-name
//SYSIN DD *
   PANVALET commands
/*
```

The name specified in DATASET is an 8-character or shorter name that becomes the member name of the object module in the HIS.OBJECT library. The PANVALET commands must include at least one ++WRITE WORK or ++INSERT WORK command. The following optional parameters can be specified on the EXEC statement:

PARM.ASM=NODECK - Suppresses storage of an object module in HIS.OBJECT.

OLIB='library' - Specifies a library other than HIS.OBJECT.

FORM='(class,,form)' - Requests special form in line printer.

Creating a Load Module

The HIS4L procedure is used to create a load module. The mainline program must reside in HIS.OBJECT in object module format (if not, use the HIS4LREP procedure described in the next section). For PL/I mainlines, submit the following job setup:

```
// JOB
// EXEC HIS4L,DATASET=object-module-name
```

The DATASET parameter specifies the name of the mainline object module in the HIS.OBJECT library. The resultant load module is stored in the HIS.LOADTST library under this same name. For assembler mainlines it is usually necessary to specify an entry point name for the load module (if you wish to specify the entry point name for a PL/I load module, code the name PLISTART):

```
// JOB
// EXEC HIS4L,DATASET=object-module-name
//SYSIN DD *
  ENTRY entry-point-name
/*
```

Subroutines stored in the automatic-call library HIS.SUBRTN4 are automatically linked with the mainline program as needed. Modules stored in any of the libraries HIS.OBJECT, HIS.LOADTST, HIS.REL4PTØ, or HIS.SUBRTN4 can be linked with the mainline as shown in the following job setup:

```
// JOB
// EXEC HIS4L,DATASET=object-module-name
//SYSIN DD *
  INCLUDE (object-module-name-1,object-module-name-2,...)
  ENTRY entry-point-name
/*
```

Modules stored in any other libraries can be linked as shown in the following job setup:

```
// JOB
// EXEC HIS4L,DATASET=object-module-name
//SYSIN DD *
  INCLUDE ddname(module-name-1,module-name-2,...)
  ENTRY entry-point-name
/*
//ddname DD ...    (for library)
```

The following optional parameters can be specified on the EXEC statement of HIS4L job setups:

OLIB='library' - Specifies an object library other than HIS.OBJECT.

PLILIB='library' - Specifies a PL/I library other than SYS1.PLIBASE.

LLIB='library' - Specifies a load library other than HIS.LOADTST.

FORM='(class,,form)' - Specifies a special form for the line printer.

LET=LET - Specifies that the load module is to be marked executable even if errors such as unresolved external references are detected.

NCAL=NCAL - (Includes LET=LET) Suppresses the automatic call function.

XREF=XREF - Requests a cross-reference listing of the load module (a load module map is printed in the absence of this option).

OVLY=OVLY - Specifies that the load module is an overlay module (one or more overlay commands must be included).

Modifying a Load Module

The HIS4LREP procedure can be used to modify an existing load module or to create a new load module. It is identical to HIS4L except that no primary input is specified. One or more INCLUDE statements must be included in the job setup. To modify an existing load module, first include all modules that are being added or replaced in the old load module and then include the old load module. For example, the following job setup can be used to modify a load module in the HIS. LOADTST library:

```
// JOB
// EXEC HIS4LREP,DATASET=load-module-name
//SYSIN DD *
  INCLUDE OBJECT(object-module-name-1,...)
  INCLUDE LOADTST(load-module-name)
  ENTRY entry-point-name
/*
```

Always specify the entry point name when modifying a load module. For PL/I load modules specify the entry point name PLISTART.

The load module produced by HIS4LREP is stored in the HIS.LOADTST library under the name specified in DATASET. All of the optional parameters shown for HIS4L in the preceding section can be used with HIS4LREP except OLIB.

Adding a Subroutine to the Automatic-Call Library

The HIS4LREP procedure can be used to place a subroutine into the HIS.SUBRTN4 automatic-call library. Submit the following job setup:

```
// JOB
// EXEC HIS4LREP,NCAL=NCAL,LLIB='HIS.SUBRTN4',DATASET=subroutine-name
//SYSIN DD *
  INCLUDE OBJECT(object-module-name)
/*
```

Placing a Program into Production Status

When a program is fully tested and ready to be assigned production status, perform the following steps:

1. Notify Highway Information System personnel of the source modules that can be given production status. They will remove the source module to the PANVALET inactive status and to the HIS backup tape.
2. Copy the finished load modules to the HIS.REL4PTØ library by re-linking using the following job setup:

   ```
   // JOB
   // EXEC HIS4LREP,LLIB='HIS.REL4PTØ',DATASET=load-module-name
   //SYSIN DD *
     INCLUDE LOADTST(load-module-name)
     ENTRY entry-point-name
   /*
   ```

   Specify PLISTART as the entry point name for PL/I load modules.
3. Delete the load modules from the HIS.LOADTST library by submitting the following job setup:

   ```
   // JOB
   // EXEC HIS
   //LOADTST DD DISP=OLD,DSNAME=HIS.LOADTST
   //SYSIN DD *
   :MODPDS
   /*
   //MODPDS DD *
   $LIBRARY=LOADTST
   load-module-name
   load-module-name
       .
       .
       .

   /*
   ```
4. Delete the object modules from the HIS.OBJECT library. Use the job setup shown above in step 3 but substitute the name OBJECT for the name LOADTST.

Commands and Instructions

The Highway Information System is a user-oriented system. Users submit commands that specify the names of application programs for execution. The commands are also used for specifying run-time options.

The HIS command format is rather difficult to process by computer. A command decoder is included as an intrinsic component of the system so that application

programs do not have to work directly with commands. The decoder translates each command into a fixed-format "instruction." An instruction is 38Ø characters in length and consists of three segments: a "system" segment, a "program" segment, and a "parm" segment.

The System Segment - The system segment of the instruction contains the application program's load module and program names. It also contains the parameters that are used only by the HIS system software. This segment cannot be easily referenced by application programs. It is 8Ø bytes in length, and has the format shown in table 1-1.

In general, the following rules apply when a parameter is decoded:

1. Character format - The field is left blank if the parameter is not coded on a command. When a parameter is coded, the specified option is stored in the field left-justified. The option is truncated on the right or padded on the right with blanks if necessary.
2. Decimal format - The field is set to decimal zero if the parameter is not coded on a command. When the parameter is coded, the option is converted to packed decimal format and stored in the instruction field.

The following parameters do not follow these standard rules:

1. TOP-MARGIN - As shown in note 1 of Table 1-1.
2. PAGE-NUMBER - As shown in note 2 of Table 1-2.
3. TABLE-NUMBER - As shown in note 3 of Table 1-3.
4. FILE and REPORT - These two parameters are equivalent. They are used to insert a subsystem number into the load module name. Recognized names are:

3Ø	ROADLOG
31	TRAFFIC
32	TRUMILE
33	ACCIDENT
34	SUFFICIENCY
35	BRIDGE
36	RAILROAD
38	URBAN-SIGN
39	SKID

5. FUNCTION - This parameter is used to insert a function number into the load module name. Recognized functions are:

TABLE 1-1

SYSTEM SEGMENT OF INSTRUCTION

Columns	Length	Format	Contents	Value When Not Coded
1	1	Char	LINKAGE	Blank
2	1	Decimal	PAGE-INCREMENT	Zero
3-4	2	Char	unused	Blank
5-12	8	Char	TITLE-DD	Blank
13-2∅	8	Char	OUTPUT-FILE	Blank
21-28	8	Char	Load module name	(Always present)
24-25	2	Char	FILE	(Always present)
24-25	2	Char	REPORT	(Always present)
28	1	Char	FUNCTION	(Always present)
29-3∅	2	Decimal	PAGESIZE	Zero
31-32	2	*Note 1	TOP-MARGIN	Zero
33-36	4	*Note 2	PAGE-NUMBER	Zero/Blank
37-38	2	*Note 3	TABLE-NUMBER	Zero
39	1	Char	PAGE-EJECT	Blank
4∅-41	2	Decimal	ODD-PAGE-POSITION	Zero
42-43	2	Decimal	EVEN-PAGE-POSITION	Zero
44-51	8	Char	PRINTER-DD	Blank
52-53	2	Decimal	COPIES	Zero
54-6∅	7	Char	unused	Blank
61-8∅	2∅	Char	Program name	(Always present)

Notes: 1. Normally decimal format. When TOP-MARGIN=NO is coded, the character string 'X ' is stored.

2. Consists of a 5-digit decimal field and a 1-byte character field. The 1-byte character field is set to:

blank	No parameter or PAGE-NUMBER=n format.
X	PAGE-NUMBER=STOP format.
$	PAGE-NUMBER=$+n format.

3. Normally decimal format. When TABLE-NUMBER=STOP is coded, the character string 'X ' is stored.

Ø DELETE
1 INSERT
2 NEW-KEY
3 REWRITE
4 SEQL-REWRITE

The Program Segment - The program segment contains parameters that specify run-time options for application programs. This segment can be referenced by calling entry point SETINST of the PRINT subroutine. The PRINT subroutine is described in chapter 13 of this manual.

The program segment is 228 characters in length, and has the format shown in Table 1-2. Standard handling of character and decimal parameters is the same as that shown above for the system segment. The following parameters do not follow these standard rules:

1. SUMMARY - A summary number is stored:

   1 RTE-NO
   2 PROJ-#
   3 COUNTY
   4 CITIES
   5 YR-BLT
   6 SUR-WD
   7 YR-IMP

2. COUNTY - The characters CO: are stored in columns 77-79 and the remainder of the field is used as a standard character field. This is done to distinguish CITY from COUNTY.

3. MAINT-DIV - This parameter is essentially a standard decimal parameter. However, the special format MAINT-DIV=ALL is allowed. A value of 99 is stored when MAINT-DIV=ALL is specified.

4. DATA - DATA utilizes 4 instruction fields:

   1. A "type" field.
   2. A "startkey" field.
   3. An "endkey" field.
   4. An "option" field.

   The option field is a standard character field which contains the actual option coded by the user. The remaining fields are filled in as shown in Table 1-3.

TABLE 1-2

PROGRAM SEGMENT OF INSTRUCTION

Columns	Length	Format	Contents	Value When Not Coded
1-3	3	Decimal	MAX-#-ENTRIES	Zero
4	1	Char	RESTART	Blank
5	1	Char	LIST	Blank
5	1	Char	MILEAGE	Blank
6-7	2	Decimal	#-ACCIDENTS	Zero
8	1	Char	DATA - Type	Blank
9-16	8	Char	SELECT-DD	Blank
17-18	2	Decimal	SQUARE-SIZE	Zero
19	1	Char	FHSUMMARY	Blank
19	1	Char	SUMMARY	Blank
2Ø	1	Char	ACCIDENTS	Blank
21-23	3	Char	FILE-NAME	Blank
24-31	8	Char	START-DATE	Blank
32-39	8	Char	END-DATE	Blank
4Ø-55	16	Char	DATA - Startkey	Blank
56-71	16	Char	DATA - Endkey	Blank
4Ø-55	16	Char	STARTKEY	Blank
56-71	16	Char	ENDKEY	Blank
46-54	9	Char	START-MILEPOINT	Blank
62-7Ø	9	Char	END-MILEPOINT	Blank
72-76	5	Char	LENGTH	Blank
77-94	18	Char	LOCATION	Blank
77-94	18	Char	CITY	Blank
77-94	18	Char	COUNTY	Blank
95-1Ø6	12	Char	START-ACCIDENT	Blank
1Ø7-118	12	Char	END-ACCIDENT	Blank
95-118	24	Char	SUMMARIES	Blank
119-126	8	Char	DDNAME	Blank
127-128	2	Decimal	#-CLUSTERS	Zero
129	1	Char	CHECKPOINT	Blank
13Ø	1	Char	DEBUG	Blank
131-138	8	Char	PASSWORD	Blank
139-14Ø	2	Decimal	CLUSTER-SKIP	Zero
141	1	Char	PTW-CONVERT	Blank
142	1	Char	SELECT-SIZE	Blank
143	1	Char	FILE-FORMAT	Blank
144	1	Char	TYPE-RUN	Blank
145-146	2	Decimal	MAINT-DIV	Zero
147	1	Char	DIRECTION	Blank
148	1	Char	LANE	Blank
149	1	Char	WHEEL	Blank
15Ø	1	Char	EFF-DATE	Blank

(Continued on following page)

TABLE 1-2 (continued)

Columns	Length	Format	Contents	Value when not coded
151-153	3	Decimal	X-COORDINATE	Zero
154-156	3	Decimal	Y-COORDINATE	Zero
157-161	5	Char	SUBSYSTEM	Blank
162-179	18	Char	DATA - Option	Blank
18Ø-181	2	Decimal	RLGTYPES	Zero
182-183	2	Decimal	TRFTYPES	Zero
184-185	2	Decimal	#-INTERSECTIONS	Zero
186	1	Char	OPTION-LIST	Blank
187	1	Char	CODE-LIST	Blank
188-19Ø	3	Decimal	SKIP	Zero
191	1	Char	FIELD-LIST	Blank
192-193	2	Decimal	#-NEW-ACC	Zero
194-196	3	Decimal	MAX-#-ACCIDENTS	Zero
197	1	Char	ACCIDENT-LEVEL	Blank
198-2Ø1	4	Char	MAINT-SECT	Blank
2Ø2-2Ø7	6	Char	START-ASSEMBLY	Blank
2Ø8-213	6	Char	END-ASSEMBLY	Blank
214-221	8	Char	DATE	Blank
222	1	Char	ACCESS	Blank
223-228	6	Char	unused	Blank

TABLE 1-3

THE DATA PARAMETER

Option	Type	Startkey	Endkey
ALL	A	AØØØØØ	Z99999
FEDERAL-AID	F	IØØØØØ	U99999
INTERSTATE+PRIMARY	C	IØØØØØ	P99999
ILOOP	L	PØØØØØ	P99999
INTERSTATE	I	IØØØØØ	I99999
PRIMARY	P	PØØØØØ	P99999
SECONDARY	S	SØØØØØ	S99999
URBAN	U	UØØØØØ	U99999
LOCAL	X	XØØØØØ	X99999
LOCAL-MUNIC	Y	YØØØØØ	Y99999
INTERSTATE=n	N	Innnnn	Innnnn
PRIMARY=n	N	Pnnnnn	Pnnnnn
SECONDARY=n	N	Snnnnn	Snnnnn
URBAN=n	N	Unnnnn	Unnnnn
LOCAL=n	N	Xnnnnn	Xnnnnn
LOCAL-MUNIC=n	N	Ynnnnn	Ynnnnn
INTERSTATE=n-m	N	Innnnn	Immmmm
PRIMARY=n-m	N	Pnnnnn	Pmmmmm
SECONDARY=n-m	N	Snnnnn	Smmmmm
URBAN=n-m	N	Unnnnn	Ummmmm
LOCAL=n-m	N	Xnnnnn	Xmmmmm
LOCAL-MUNIC=n-m	N	Ynnnnn	Ymmmmm

Notes:
1. In the absence of START-MILEPOINT, blanks are stored in the milepoint portion of startkey.
2. In the absence of END-MILEPOINT, '999999999' is stored in the milepoint portion of endkey.

The Parm Segment - The parm segment is used for passing parameters to an application program following standard system/370 OS/VS1 conventions. The segment has been implemented for the following reasons:

1. Programs that were not originally designed to run under HIS can be executed under HIS.
2. Execution-time options such as ISASIZE can be passed to PL/I Optimizer programs.

The parm segment is 72 bytes in length. The first two bytes specifies the number of bytes in the parameter. The remainder is the parameter (left-justified with trailing blanks).

Cataloging a Program Name

Program names for use on a command are kept in the program table. A description of the table and its maintenance is included in chapter 11 of this manual.

Cataloging an Execution-Time Parameter

It is possible to catalog a parameter which will be passed to an application program in the parm segment whenever the program is run. The cataloged parameters are kept in the passed parameter table. A description of the table and its maintenance is included in chapter 11 of this manual.

Naming Conventions

Source modules and load modules of the Highway Information System have been given names of the following formats:

1-3	'HIS'
4-5	Subsystem number
6-8	Program number within subsystem

The subsystem numbers are:

20	Supervisory modules and miscellaneous subroutines
21	Tables subsystem
22	Select subsystem
23	Utility programs subsystem
30	Roadlog subsystem
31	Traffic subsystem
32	True mileage subsystem
33	Accident subsystem
34	Sufficiency subsystem
35	Bridge subsystem
36	Railroad subsystem
37	Grid table subsystem
38	Urban sign subsystem
39	Skid subsystem
40	Open range subsystem
41	EMS subsystem

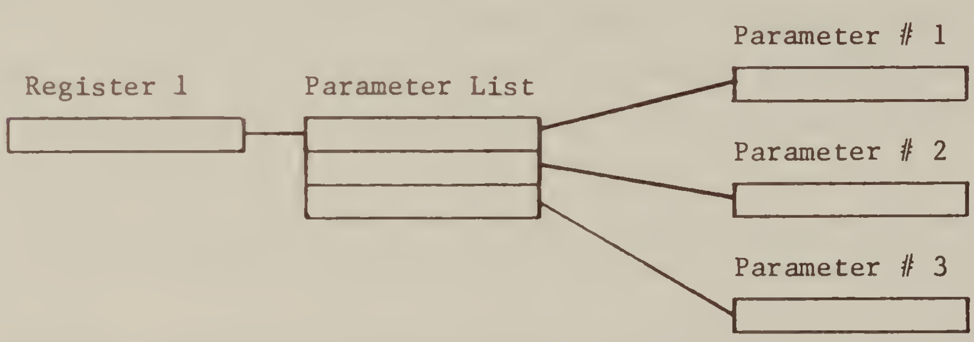
PL/I and Assembler Linkage Conventions

This chapter describes the linkage conventions used with assembler and PL/I routines. These conventions must be used when combining the two languages. The conventions described here apply to both the PL/I (F) compiler and the PL/I Optimizing compiler.

Register Usage - OS/VS sets a standard set of conventions for the following general registers:

Ø	Argument linkage
1	Argument linkage
13	Savearea linkage
14	Subroutine linkage
15	Entry point address and return code

Register Ø is generally used only by IBM system routines and by some supervisor and data management macros for argument linkage. Register 1 is the more commonly used register for argument linkage. A parameter list is built, containing the fullword address for each argument passed. When a subroutine is called, the address of the parameter list is loaded into register 1. The following diagram illustrates the use of register 1 to pass three arguments:



The standard conventions require that the last word of the parameter list contains a 1 is the first bit and that all other words contain a Ø is the first bit. The HIS routines do not test for the presence of this "last parameter" indicator, and it is not required.

When a subroutine is called, register 13 must contain the address of an 18-word area (a "savearea") in which the general registers can be saved. The format of a savearea is:

Word	Contents
1	Used only by PL/I
2	Address of previous save area
3	Address of next save area
4	R-14
5	R-15
6	R-Ø
7	R-1
8	R-2
9	R-3
1Ø	R-4
11	R-5
12	R-6
13	R-7
14	R-8
15	R-9
16	R-1Ø
17	R-11
18	R-12

Upon entry to any routine, the routine must first save the registers in the provided save area. If any system macros will be used or if any subroutines will be called. the routine must establish a savearea for use by the called routines.

PL/I expands upon the savearea concept. Instead of passing simply a savearea, PL/I routines pass a Dynamic Storage Area (DSA). The first 18 words of a DSA form a normal savearea. When an assembler program is called by a PL/I program, the DSA can be used as a normal savearea. The assembler program must take care not to alter the first word of the DSA, or the PL/I program may fail.

Register 14 is used for subroutine linkage. Upon entry to any routine, this register contains the address to which control is to be returned upon completion.

Upon entry to any routine, register 15 must contain the entry point address. Upon return from a routine, register 15 is used as a return code.

PL/I uses register 12 as a dedicated register containing the address of a Task Communications Area (TCA). An assembler program that is called by a PL/I program must take care not to alter this register at any time unless both a STAE and a SPIE macro are issued to prevent the PL/I routines from taking control during an ABEND or program interrupt condition.

Subroutine linkages performed by PL/I are rather elaborate, and are difficult to use when combining PL/I and assembler programs. Assembler programs that are called by PL/I programs and that do not call other PL/I programs can by and large ignore the PL/I conventions and follow standard assembler conventions. If an assembler program calls another PL/I program, however, the linkages can become more difficult. The assembler program must do one of the following:

1. Link into the DSA chains exactly as a PL/I routine does.
2. Become invisibile to PL/I by not linking into any of the chains.

The second alternative is by far the easiest. Simply save the registers in an area within the program rather than in the passed savearea, and leave register 13 unchanged.

PL/I Arguments - The following data types are used as arguments in the Highway Information System between PL/I and assembler programs:

```
BINARY FIXED
DECIMAL FIXED
CHARACTER
POINTER
```

A BINARY FIXED variable requires a halfword for 1-15 bits or a fullword for 16-31 bits. BINARY variables are passed following the standard conventions. Consider the following sample program:

```
/* SAMPLE PROGRAM - BINARY FIXED LINKAGE */
DECLARE BINA BINARY FIXED(15),
        BINB BINARY FIXED(31);
CALL SUBRTN (BINA,BINB);
```

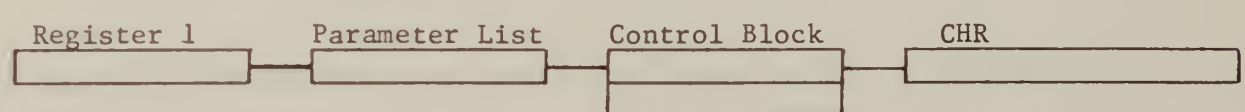
The following diagram shows the linkage upon entry to SUBRTN:

DECIMAL FIXED variables are stored in standard packed-decimal format. They are passed directly as shown above for BINARY FIXED variables. BINARY FLOATING and DECIMAL FLOATING variables are also passed directly (these types are not used extensively in HIS).

A pointer is simply a fullword address. A null pointer has the value X'FFØØØØØØ'. Pointers are passed directly as shown in the following example:

```
/* SAMPLE PROGRAM - POINTER ARGUMENTS */
DECLARE PTRA POINTER,
        CHR  CHAR(8);
PTRA = ADDR(CHR);
CALL SUBRTN (PTRA);
```

Fixed-length character strings are stored one-character per byte in the standard character format. When a string is passed, an intermediate control block is passed. The F compiler and the Optimizing compiler have different formats for this block, but in both cases the block is 8 bytes long and the first word of the block contains the address of the character string. The linkages are as shown in this example:



Because the second word of the control block is different for the two compilers, the HIS routines utilize only the first word. The linkage becomes identical to linkage via a pointer variable.

Variable-length character strings are handled identically to fixed-length character strings when using the F compiler. When using the Optimizing compiler, the control block points to a halfword that precedes the character string and that contains the current length of the string.

Other PL/I data types such as arrays, structures, files, and statement labels are passed with complicated control blocks. The formats of these blocks vary considerably between the Optimizing compiler and the F compiler. Within HIS, these data types are not passed between PL/I and assembler programs. It is much more convenient to pass pointers that provide needed addresses.

The formats of PL/I (F) control blocks are described in the IBM publication PL/I Subroutine Library PLM. The Optimizing compiler's control blocks are described in PL/I Optimizing Compiler: Execution Logic.

Programming Techniques and Facilities for HIS

This section provides a brief introduction to the subroutines and facilities available to programmers writing programs to run in the Highway Information System.

Printed Output - The PRINT subroutine described in chapter 13 should be used for all printed output produced under HIS, although separate files can be used where necessary. The PRINT subroutine is easy to use and provides several facilities such as page headings and automatic advance to a new page when necessary that normally require extra programming in each application program. The subroutine also provides users with a number of formatting functions.

Checking for DD Statements - If a program attempts to open a file when no DD statement has been included with a run, the program will fail and the OS/VS1 operating system does not provide an adequate error message for the condition. The user is aided immensely if each program checks to be sure all of its DD statements are present before opening its files. The subroutines CHECKDD and DUMPDD described in chapter 13 can be used for this purpose.

Date Routines - A number of date subroutines are provided. These routines can calculate julian date or day of week from a given date, can return the current date, can edit a date to be sure it is valid, and can convert a date to different formats. These routines are described in chapter 13.

File Access Subroutines - File access subroutines are provided for all major Highway Information System files. These subroutines greatly decrease the amount of programming time needed to write and debug a program. Most of the file have two levels of sequential read subroutines: a "low-level" subroutine and a "high-level" subroutine. The low-level routines provide all of the basic access functions. The high-level routines include a number of higher-level functions controlled directly by the HIS command, such as the SELECT facilities. Access subroutines are provided for some of the files for direct reads, sequential loads, and updates.

Tables - The use of tables greatly increases the flexibility and maintainability of computer programs. The use of disk-resident tables rather than compiled tables increases the flexibility even further because the tables can be altered without re-compiling programs. The library HIS.TABLES has been included as a basic component of the Highway Information System for the purpose of storing such tables. Several utility programs are provided in the system for loading and maintaining the tables. Several access subroutines are provided to allow programs to easily retrieve a table. The tables software is described in chapter 11.

Dynamic Subroutines and Interfaces - A number of often-used subroutines have been implemented as dynamic subroutines. The subroutine is link-edited as a separate load module. An interface subroutine is link-edited with the calling program. The interface subroutine retrieves the actual subroutine whenever it is needed. The actual subroutine resides in only one load module so that only one link-edit is required when the subroutine is modified. To write a dynamic subroutine, follow the following guidelines:

1. Dynamic subroutines must be written in assembler language. The interface subroutine must also be written in assembler language.
2. If the subroutine contains more than one entry point, add an entry point that is an "address list." At this entry point, store the address of each entry point. The address list becomes the entry point address of the load module.

3. If the subroutine contains only one entry point, no address list is needed. The subroutine's entry point becomes the load module's entry point.
4. Write an interface subroutine that calls the FETCH subroutine. Pass the subroutine name rather than the actual load module name to FETCH so that the interface does not become tied to a set of name conventions. The interface subroutine is linked with all calling programs.
5. Add an entry to the equivalence table (see chapter 11) that relates the subroutine name to a load module name.

THE ACCIDENT FILES

File Description - Accident Detail File

The format of accident detail records is:

Columns	Length	Format	Item	Data Element
1	1	char(1)		Delete byte
2-13	12	char(12)	A-1	Key
14-15	2	fixed(2)	A-2	Date occurred - month
16-17	2	fixed(2)	A-2	Date occurred - day
18-19	2	fixed(2)	A-2	Date occurred - year
20-21	2	fixed(2)	A-2	Time occurred - hour
22-23	2	fixed(2)	A-2	Time occurred - minute
24-25	2	fixed(2)	B-2	Date notified - month
26-27	2	fixed(2)	B-2	Date notified - day
28-29	2	fixed(2)	B-2	Date notified - year
30-31	2	fixed(2)	B-2	Time notified - hour
32-33	2	fixed(2)	B-2	Time notified - minute
34-35	2	fixed(2)	B-3	Date arrived - month
36-37	2	fixed(2)	B-3	Date arrived - day
38-39	2	fixed(2)	B-3	Date arrived - year
40-41	2	fixed(2)	B-3	Time arrived - hour
42-43	2	fixed(2)	B-3	Time arrived - minute
44-45	2	fixed(3)	A-3	City number
46-47	2	fixed(2)	A-4	County number
48-59	12	char(12)	A-5	Location
60-61	2	fixed(2)	A-6	First harmful event
62-63	2	fixed(2)	A-7	First object hit off roadway
64	1	fixed(1)	A-8	Injury severity
65	1	fixed(1)	A-9	Damage severity
66	1	fixed(1)	A-10	Class of trafficway
67	1	fixed(1)	A-11	Roadway-related location
68	1	fixed(1)	A-12	Junction-related location
69-70	2	fixed(2)	A-13	Number of vehicles
71-72	2	fixed(2)	A-14	Number of pedestrians
73-74	2	fixed(2)	A-15	Number of fatalities
75-76	2	fixed(2)	A-16	Number of injuries
77	1	fixed(1)	A-17	Weather condition
78	1	fixed(1)	A-18	Road condition
79	1	fixed(1)	A-19	Light condition
80-81	2	fixed(2)	A-20	Traffic controls
82-83	2	fixed(2)	A-21	Other damage - type
84	1	fixed(1)	A-22	Other damage - severity
85	1	fixed(1)	A-23	Other damage - owner
86-87	2	fixed(2)	A-24	Posted speed limit
88	1	char(1)	A-25	Engineering study
89-90	2	fixed(2)	A-26	Analysis - field 1
91-92	2	fixed(2)	A-26	Analysis - field 2
93	1	fixed(1)	A-27	Collision type
94	1	char(1)		Reportable
95	1	char(1)		Investigated
96	1			Unused at this time

The contents of these data elements are described in the publication <u>Highway Information System Release 4.Ø - Data Coding Manual</u>. The item numbers listed under the column "Item" are the item numbers used in that manual.

All data elements are stored in standard IBM character or packed decimal (fixed) formats. For character formats, the length specification (as in char(12)) gives the length in characters. For fixed formats, the length specification gives the length in digits.

When a record is deleted from the file, it is not physically removed until the file is reorganized. The delete byte in column one is used by the system to indicate deleted records so that they can be ignored.

The reportable code in column 94 is inserted by the update program. It contains one of these codes:

X	Legally reportable
blank	Not reportable

The investigated code in column 95 is inserted by the update program. It contains one of these codes:

blank	Accident was investigated
X	Accident was not investigated

<u>File Description - Accident Vehicle File</u>

The format of accident vehicle records is:

Columns	Length	Format	Item	Data Element
1	1	char(1)		Delete byte
2-13	12	char(12)	A-1	Accident number
14	1	char(1)		Vehicle/pedestrian
15-16	2	char(2)		Vehicle/pedestrian number
17-38	22	char(22)	C-2	Name
39-55	17	char(17)	C-3	Driver license number
56-57	2	char(2)	C-4	Driver license state
58-63	6	char(6)	C-5	Date of birth
64	1	char(1)	C-6	Re-examination code
65-7Ø	6	char(6)	C-7	Charge code
71-76	6	char(6)	C-8	Summons number
77	1	fixed(1)	D-2	Contr. circ. - vision
78	1	fixed(1)	D-2	Contr. circ. - physical defects
79	1	fixed(1)	D-2	Contr. circ. - road defects
8Ø	1	fixed(1)	D-2	Contr. circ. - mechanical defects
81	1	fixed(1)	D-2	Contr. circ. - possible violations

Columns	Length	Format	Item	Data Element
82	1	fixed(1)	D-3	Driver - alcohol
83	1	char(1)	D-3	Driver - sex
84	1	fixed(1)	D-3	Driver - injury severity
85-86	2	fixed(2)	D-3	Driver - Age
87	1	fixed(1)	D-3	Passenger fc - alcohol
88	1	char(1)	D-3	Passenger fc - sex
89	1	fixed(1)	D-3	Passenger fc - injury severity
90-91	2	fixed(2)	D-3	Passenger fc - age
92	1	fixed(1)	D-3	Passenger fr - alcohol
93	1	char(1)	D-3	Passenger fr - sex
94	1	fixed(1)	D-3	Passenger fr - injury severity
95-96	2	fixed(2)	D-3	Passenger fr - age
97	1	fixed(1)	D-3	Passenger rl - alcohol
98	1	char(1)	D-3	Passenger rl - sex
99	1	fixed(1)	D-3	Passenger rl - injury severity
100-101	2	fixed(2)	D-3	Passenger rl - age
102	1	fixed(1)	D-3	Passenger rc - alcohol
103	1	char(1)	D-3	Passenger rc - sex
104	1	fixed(1)	D-3	Passenger rc - injury severity
105-106	2	fixed(2)	D-3	Passenger rc - age
107	1	fixed(1)	D-3	Passenger rr - alcohol
108	1	char(1)	D-3	Passenger rr - sex
109	1	fixed(1)	D-3	Passenger rr - injury severity
110-111	2	fixed(2)	D-3	Passenger rr - age
112-113	2	fixed(2)	D-10	Vehicle year
114-115	2	fixed(2)	D-5,7	Vehicle/pedestrian intent
116-117	2	fixed(2)	D-8	Body style
118	1	fixed(1)	D-9	Trailer style
119	1	char(1)	D-11	Vehicle involved in interstate traffic
120-134	15	char(15)	D-12	Vehicle license number or VIN
135	1	char(1)	D-13	Damage level
136	1	fixed(1)	D-14	Damage severity

The contents of these data elements are described in the publication Highway Information System Release 4.0 - Data Coding Manual. The item numbers listed under the column "Item" are the item numbers used in that manual.

All data elements are stored in standard IBM character or packed decimal (fixed) formats.

When a record is deleted from the file, it is not physically removed until the file is reorganized. The delete byte in column is used by the system to indicate delete records so that they can be ignored.

File Description - Accident Directory File

The format of accident directory records is:

Columns	Length	Format	Contents
1	1	char(1)	Delete byte
2-16	15	char(15)	Key
17-28	12	char(12)	Accident number
29-30	2	fixed(2)	Number of fatalities
31-32	2	fixed(2)	Number of injuries
33-34	2	fixed(2)	Date occurred - month
35-36	2	fixed(2)	Date occurred - day
37-38	2	fixed(2)	Date occurred - year
39-40	2	fixed(2)	Time occurred - hour
41-42	2	fixed(2)	First harmful event
43	1	fixed(1)	Collision type
44	1	fixed(1)	Road condition
45	1	fixed(1)	Number of lanes
46	1		Unused at this time

This format is the record format used by all programs in the system that read records for processing. The records are actually stored in slightly different format (the key is stored using only the low-order three digits of the route number, shortening the key to 13 characters and the record to 44 characters). The records are reformatted for processing programs by the file access subroutines.

The directory file is generated periodically from the accident detail file and from the roadlog file. The number of lanes field is retrieved from the roadlog file. All other data elements come from the accident detail file. The key is obtained from the location field with minor reformatting to match the key format used in other files.

Because several accidents can occur at the same location, the 15-character key made up of route system, route number, and milepoint is not adequate to provide a unique key for each accident. The file key, therefore, consists of both the 15-character milepoint key plus the 12-character accident number.

File Description - Accident-by-Sections File

The format of accident-by-sections records is:

Columns	Length	Format	Contents
1	1	char(1)	Delete byte
2-16	15	char(15)	Key at beginning of section
17	1	char(1)	Remark
18-32	15	char(15)	Key at end of section
33-67	35	char(35)	Description
68-71	4	fixed(7,3)	Section length
72-73	2	fixed(2)	Oldest year - year
74-77	4	fixed(7)	Oldest year - Adjusted vehicle miles
78-81	4	fixed(7,3)	Oldest year - Adjusted section length
82-83	2	fixed(3)	Oldest year - Number of accidents
84-85	2	fixed(3)	Oldest year - Number of injury accidents
86-87	2	fixed(3)	Oldest year - Number of fatal accidents
88-89	2	fixed(3)	Oldest year - Number of injuries
90-91	2	fixed(3)	Oldest year - Number of fatalities
92-93	2	fixed(2)	Second year - Year
94-97	4	fixed(7)	Second year - Adjusted vehicle miles
98-101	4	fixed(7,3)	Second year - Adjusted section length
102-103	2	fixed(3)	Second year - Number of accidents
104-105	2	fixed(3)	Second year - Number of injury accidents
106-107	2	fixed(3)	Second year - Number of fatal accidents
108-109	2	fixed(3)	Second year - Number of injuries
110-111	2	fixed(3)	Second year - Number of fatalities
112-113	2	fixed(2)	Third year - Year
114-117	4	fixed(7)	Third year - Adjusted vehicle miles
118-121	4	fixed(7,3)	Third year - Adjusted section length
122-123	2	fixed(3)	Third year - Number of accidents
124-125	2	fixed(3)	Third year - Number of injury accidents
126-127	2	fixed(3)	Third year - Number of fatal accidents
128-129	2	fixed(3)	Third year - Number of injuries
130-131	2	fixed(3)	Third year - Number of fatalities
132-133	2	fixed(2)	Earliest effective date - month
134-135	2	fixed(2)	Earliest effective date - day
136-137	2	fixed(2)	Earliest effective date - year
138-139	2	fixed(2)	Latest effective date - month
140-141	2	fixed(2)	Latest effective date - day
142-143	2	fixed(2)	Latest effective date - year
144	1	fixed(1)	Number of lanes
145-146	2	fixed(3)	City number

A totals record (remark code X) is stored for each route with a key of snnnnn999+0.000 (snnnnn is the system and route number). In addition, a totals record is stored for each route system with a key of s99999999+0.000. The format of the remarks records is:

Columns	Length	Format	Contents
1	1	char(1)	Delete byte
2-16	15	char(15)	Key
17	1	char(1)	Remark (contains X)
18-22	5	fixed(9,3)	Total section length
23-24	2	fixed(2)	Oldest year - Year
25-30	6	fixed(11)	Oldest year - Adjusted vehicle miles
31-35	5	fixed(9,3)	Oldest year - Adjusted section length
36-39	4	fixed(7)	Oldest year - Number of accidents
40-43	4	fixed(7)	Oldest year - Number of injury accidents
44-47	4	fixed(7)	Oldest year - Number of fatal accidents
48-51	4	fixed(7)	Oldest year - Number of injuries
52-55	4	fixed(7)	Oldest year - Number of fatalities
56-57	2	fixed(2)	Second year - Year
58-63	6	fixed(11)	Second year - Adjusted vehicle miles
64-68	5	fixed(9,3)	Second year - Adjusted section length
69-72	4	fixed(7)	Second year - Number of accidents
73-76	4	fixed(7)	Second year - Number of injury accidents
77-80	4	fixed(7)	Second year - Number of fatal accidents
81-84	4	fixed(7)	Second year - Number of injuries
85-88	4	fixed(7)	Second year - Number of fatalities
89-90	2	fixed(2)	Third year - Year
91-96	6	fixed(11)	Third year - Adjusted vehicle miles
97-101	5	fixed(9,3)	Third year - Adjusted section length
102-105	4	fixed(7)	Third year - Number of accidents
106-109	4	fixed(7)	Third year - Number of injury accidents
110-113	4	fixed(7)	Third year - Number of fatal accidents
114-117	4	fixed(7)	Third year - Number of injuries
118-121	4	fixed(7)	Third year - Number of fatalities
122	1	char(1)	ACCIDENT-LEVEL parameter
123-140	18	char(18)	DATA parameter
141-146	6		Unused at this time

The ACCIDENT-LEVEL and DATA parameters stored are the parameters that were specified when the file was generated.

File Description - Intersection Directory File

This file is a scratch file used by the HIGH-ACC-INTERSECTN program. The record format is:

Columns	Length	Format	Contents
1	1	char(1)	Delete byte
2-5	4	char(4)	Intersection number
6-17	12	char(12)	Accident number
18-19	2	fixed(2)	Date occurred - month
20-21	2	fixed(2)	Date occurred - day
22-23	2	fixed(2)	Date occurred - year
24-25	2	fixed(2)	Time occurred - hour
26-27	2	fixed(2)	Time occurred - minute
28-30	3	fixed(4)	X coordinate
31-33	3	fixed(4)	Y coordinate
34-35	2	fixed(2)	First harmful event
36	1	fixed(1)	Junction-related location
37-38	2	fixed(2)	Number of vehicles
39-40	2	fixed(2)	Number of pedestrians
41-42	2	fixed(2)	Number of fatalities
43-44	2	fixed(2)	Number of injuries
45	1	fixed(1)	Weather condition
46	1	fixed(1)	Road condition
47	1	fixed(1)	Light condition
48-49	2	fixed(2)	Traffic controls
50	1	fixed(1)	Collision type

File Description - Intersection Index File

This file is a scratch file used by the HIGH-ACC-INTERSECTN program. The record format is:

Columns	Length	Format	Contents
1-4	4	char(4)	Intersection number
5-8	4	char(4)	Number of accidents
9-18	10		Unused at this time

File Description - Memos File

The record format of the memos file is:

Columns	Length	Format	Contents
1-12	12	char(12)	Accident number
13-14	2	char(2)	Date occurred - month
15-16	2	char(2)	Date occurred - day
17-18	2	char(2)	Date occurred - year
19-2Ø	2	fixed(2)	County number
21-32	12	char(12)	Accident location
33-34	2	char(2)	Number of vehicles
35	1	char(1)	Injury severity
36-57	22	char(22)	Name
58-74	17	char(17)	Driver license number
75-76	2	char(2)	Driver license state
77-82	6	char(6)	Birth date
83-88	6	char(6)	Charge code
89-9Ø	2	fixed(3)	City number

The DACRDQ Subroutine

DACRDQ is a "low-level" subroutine for reading the accident detail file sequentially. It provides the basic access functions, but not the higher-level functions such as the select capability. Programs that summarize accident data should utilize the ACCRD subroutine to implement the higher-level features.

DACRDQ has the following entry points:

DACRDQF	Initialization - open the file
DACRDQR	Set startkey-endkey range
DACRDQ	Sequential read
DACRDQX	Direct read - key-high
DACRDQP	Position - key-high
DACRDQC	Close the file
DACRDQI	Combines functions of DACRDQF and DACRDQR

DACRDQF Entry Point - Call DACRDQF to open the file. Pass a binary fixed(15) value of 1.

DACRDQR Entry Point - The calling program can limit access to a portion of the file by calling DACRDQR. Pass a char(12) starting accident number and a char(12) ending accident number. After the call, the program has access to only those accidents having a number equal to or between the starting and ending numbers passed.

DACRDQ Entry Point - Call DACRDQ to read a record. Pass a pointer variable. The subroutine reads a record and places its address in the pointer. At end-of-file, a null pointer is returned.

DACRDQX Entry Point - DACRDQX allows access to the file by accident number. Pass a char(12) accident number and a pointer. The subroutine retrieves the first accident in the file having a number equal to or larger than the passed number and returns its address in the pointer. If the passed number is larger than that of the last accident in the file, a null pointer is returned.

DACRDQP Entry Point - Call DACRDQP to position key-high to an accident. Pass a char(12) accident number. A call to DACRDQP followed by a call to DACRDQ is equivalent to a single call to DACRDQX.

DACRDQC Entry Point - Call DACRDQC to close the file. No arguments are passed.

DACRDQI Entry Point - This entry point combines the functions of DACRDQF and DACRDQR. Pass a binary fixed(15) value of 1, a char(12) starting accident number, and a char(12) ending accident number.

Error Messages - If the calling program fails to open the file or if a value other than 1 is passed when the file is opened, the message

    ***** DACRDQ-Ø1 - DACRDQ NOT PROPERLY INITIALIZED - ENTRY POINT name

is printed and a user-1ØØ abend is issued.

The DACRDB Subroutine

DACRDB is a subroutine for reading the accident detail file directly. An accident number must be specified for each call.

DACRDB has these entry points:

DACRDBF	Initialization - open the file
DACRDB	Direct read - key-equal
DACRDBC	Close the file

DACRDBF Entry Point - Call DACRDBF to open the file. Pass a binary fixed(15) value of 1.

DACRDB Entry Point - Call DACRDB to read a record. Pass a pointer and a char(12) accident number. The specified record is read and its address returned in the pointer. A null pointer is returned if the record does not exist.

DACRDBC Entry Point - Call DACRDBC to close the file. No arguments are passed.

Error Messages - If an input/output error occurs, the message

***** DACRDB-Ø1 - I/O ERROR IN DACRDB *****

is printed and a user-1ØØ abend is issued.

The VACRDQ Subroutine

VACRDQ is a "low-level" subroutine for reading the accident vehicle file sequentially. Programs that summarize accident data should utilize the ACCRD subroutine rather than VACRDQ in order to implement the higher-level features.

VACRDQ has these entry points:

VACRDQF	Initialization - open the file
VACRDQR	Set startkey-endkey range
VACRDQ	Sequential read
VACRDQX	Direct read - key-high
VACRDQP	Position - key-high
VACRDQC	Close the file
VACRDQI	Combines functions of VACRDQF and VACRDQR.

Use of VACRDQ is similar to DACRDQ - see above. Accident numbers passed to VACRDQR, VACRDQX, VACRDQP, and VACRDQI are passed in char(15) format instead of in char(12) format because of the longer key of the accident vehicle file.

The VACRDA Subroutine

VACRDA reads all of the records in the accident vehicle file for a specified accident. Entry points are:

VACRDAI	Initialization - open the file
VACRDAR	Set startkey-endkey range
VACRDA	Direct read - key-equal
VACRDAS	Sequential read
VACRDAC	Close the file

VACRDAI Entry Point - Call VACRDAI to open the file. Pass a binary fixed(15) value of 1.

VACRDAR Entry Point - VACRDAR can be called to set a startkey-endkey range. Pass a char(15) starting key and a char(15) ending key. After the call, the program has access to only that portion of the file lying between the two keys (inclusive).

VACRDA Entry Point - Call VACRDA to read direct. Pass a char(12) accident number, a pointer, and a binary fixed(15) variable. The vehicle records for the accident are read into an array and the array address is stored in the pointer. A count of the number of records read is returned in the binary variable. If no records are stored for the passed number, a null pointer and a count of zero are returned.

VACRDAS Entry Point - Call VACRDAS to read sequentially. Pass a pointer and a binary fixed(15) variable. The pointer and binary variable are used as with a VACRDA call. A null pointer and a count of zero are returned at end-of-file.

VACRDAC Entry Point - Call VACRDAC to close the file. No arguments are passed.

Error Messages - If the calling program fails to call VACRDQI or passes a value other than 1, the message

***** VACRDA-Ø1 - VACRDA NOT PROPERLY INITIALIZED *****

is printed and a user-1ØØ abend is issued.

The ACDRDQ Subroutine

ACDRDQ is a low-level subroutine for reading the accident directory file sequentially. The ACCRD subroutine should be used if higher-level functions are being implemented.

ACDRDQ entry points are:

ACDRDQF	Initialization - open the file
ACDRDQR	Set startkey-endkey range
ACDRDQ	Sequential read
ACDRDQX	Direct read - key-high
ACDRDQP	Position - key-high
ACDRDQC	Close the file
ACDRDQI	Combines functions of ACDRDQF and ACDRDQR

Use of these subroutine is similar to DACRDQ, described above. Pass keys in char(15) format (route system, route number, and milepoint).

The ACDWRQ Subroutine

ACDWRQ is a subroutine for loading the accident directory file. Entry points are:

ACDWRQ	Write a record
ACDWRQC	Close the file

ACDWRQ Entry Point - Call ACDWRQ to write a record. Pass a pointer that contains the record's address. Records must be written in order by key.

ACDWRQC Entry Point - Call ACDWRQC to close the file. No arguments are passed.

The ACCRD Subroutine

ACCRD is a "high-level" subroutine for reading the accident directory, accident detail, and/or accident vehicle files. It should be used by programs that print listings or summaries of accident data so that the select capabilities will be implemented. Note that only one high-level subroutine can be called by any one program.

ACCRD automatically handles the following parameters from HIS commands without intervention from the calling program:

MAX-#-ENTRIES
START-DATE and END-DATE
START-ACCIDENT and END-ACCIDENT
LOCATION
CITY
COUNTY
DATA, START-MILEPOINT, and END-MILEPOINT
SELECT-DD and SELECT-SIZE

All of these parameters are optional if the calling program utilizes only the detail and vehicle file. A DATA parameter is required if the calling program utilizes the directory file.

ACCRD has the following entry points:

ACCRDF	Initialization - open the file(s)
ACCRDO	Retrieve location type
ACCRD	Sequential read
ACCRDC	Close the file(s)

ACCRDF Entry Point - Call ACCRDF to open the file(s). Pass three binary fixed(15) value. Pass a value of 2 in the first and a value of 1 in the second. The third variable indicates the file(s) that the calling program requires to be read:

1	Directory file
2	Detail file
3	Directory and detail file
4	Vehicle file
5	Directory and vehicle file
6	Detail and vehicle file
7	All three files

The subroutine opens the files requested by the calling program and may open additional files if needed for processing the parameters specified in the command. If a DATA parameter is specified, the directory file is opened. (Note that if the calling program passes an odd number, a DATA parameter is required on the command). If LOCATION, CITY, OR COUNTY is specified, the detail file is opened. If the calling program requests only the vehicle file, the detail file is opened if START-DATE or END-DATE is specified.

ACCRDO Entry Point - The calling program can determine which (if any) of the parameters LOCATION, CITY, or COUNTY was specified on the command by calling ACCRDO. Pass a binary fixed(15) variable. Upon return, the variable will contain one of these values:

Ø	Invalid parameter
1	No parameter
2	LOCATION=STATEWIDE
3	CITY parameter
4	COUNTY parameter
5	LOCATION=EVERYTHING

ACCRD Entry Point - Call ACCRD to read the record(s) for an accident. Pass the following arguments:

Directory pointer
Detail pointer
Vehicle pointer
Number of vehicle records - binary fixed(15)

After an ACCRD call, these arguments will have these results:

Directory pointer - If directory records are being read, the address of a directory record. At end-of-file or if directory records are not being read, a null value.

Detail pointer - If detail records are being read, the address of a detail record. At end-of-file or if directory records are not being read, a null value.

Vehicle pointer - If vehicle records are being read, the address of an array containing the vehicle record(s). At end-of-file or if vehicle records are not being read, a null value.

Number of vehicle records - If vehicle records are being read, the number of vehicle records in the array. At end-of-file or if vehicle records are not being read, a value of zero.

ACCRDC Entry Point - Call ACCRDC to close the files. No arguments are passed.

Error Messages - If an error is detected, an appropriate message is printed. A user-1ØØ abend is issued for all messages except ACCRD-1ØØ.

***** ACCRD-Ø1 - ACCRD NOT PROPERLY INITIALIZED - ENTRY POINT name

Entry point ACCRDF was not called.

***** ACCRD-Ø2 - NO VEHICLE RECORDS FOR ACCIDENT accident-number

The vehicle file did not contain any records for the printed accident number. The accident is not processed.

***** ACCRD-Ø3 - NO DETAIL RECORD FOR ACCIDENT accident-number

The detail file did not contain a record for the printed accident number. The accident is not processed.

***** ACCRD-Ø4 - INVALID LOCATION, CITY, OR COUNTY - parameter

The name specified in a LOCATION, CITY, or COUNTY parameter is not known.

***** ACCRD-Ø5 - DATA PARAMETER MISSING

No DATA parameter was coded on the command but the calling program attempted to read the directory file.

***** ACCRD-1ØØ - MAX-#-ENTRIES COMPLETED

This message is a warning message to indicate that processing was terminated due to a MAX-#-ENTRIES parameter.

Sample Program - The following sample program illustrates the use of ACCRD:

```
/* SAMPLE PROGRAM - USE OF ACCRD SUBROUTINE */
SAMPLE:  PROCEDURE OPTIONS (MAIN);
DECLARE ACCRDF ENTRY (BIN FIXED,BIN FIXED,BIN FIXED),
        ACCRD  ENTRY (POINTER,POINTER,POINTER,BIN FIXED),
        ACCRDC ENTRY,
        DIRPTR POINTER,
        DETPTR POINTER,
        1  DET BASED(DETPTR),
           2  DELETEBYTE CHAR(1),
           2  ACC#       CHAR(12),
                 .
                 .
                 .
        VEHPTR POINTER,
        #VEH   BIN FIXED;

        CALL ACCRDF (2,1,2);   /* DETAIL FILE ONLY */
LOOP:   CALL ACCRD (DIRPTR,DETPTR,VEHPTR,#VEH);
        IF DETPTR=NULL() THEN GOTO TERMINATE;
                .
                .
                .
        GOTO LOOP;

TERMINATE:
        CALL ACCRDC;
        END SAMPLE;
```

In this sample program, VEHPTR will always be returned with a null value and #VEH will always be returned with a zero value. DIRPTR will be returned with a null value if no DATA parameter is coded on the command, but will contain the address of a directory record if a DATA parameter is coded. In the absence of a DATA parameter, the detail records will be returned in order by accident number. When a DATA parameter is coded, the detail records are returned in order by milepoint.

The CVTACC Subroutine

CVTACC can convert accident data cards into accident records and vice versa. Entry points are:

CVTDETA	Convert an A-B card combination to detail record format
CVTDETB	Convert a detail record to an A-B card combination
CVTVEHA	Convert a C-D card combination to vehicle record format
CVTVEHB	Convert a vehicle record to a C-D card combination

The name CVTACC is not an entry point to the subroutine, but the subroutine is linked by automatic call using the name CVTACC. Hence, the name CVTACC should be declared as an entry point even though it is not called.

CVTDETA Entry Point - Call CVTDETA to convert an A-B card combination into a detail record. Pass the char(8Ø) A card, the char(8Ø) B card, and the char(96) detail record. The reportable code may not be set correctly after the call because it depends upon information specified in D cards - the detail record is again passed to each CVTVEHA call to complete the setup of this code.

CVTDETB Entry Point - Call CVTDETB to convert a detail record into A-B card format. Pass the char(8Ø) A card, the char(8Ø) B card, and the char(96) detail record.

CVTVEHA Entry Point - Call CVTVEHA to convert a C-D card combination to vehicle record format. Pass the char(8Ø) C card, the char(8Ø) D card, the char(136) vehicle record, and the char(96) detail record.

CVTVEHB Entry Point - Call CVTVEHB to convert a record into a C-D card. Pass the char(8Ø) C card, the char(8Ø) D card, the char(136) vehicle record, and the char(96) detail record.

Data cards must be edited for numerics only in numeric fields before converting to record format. The presence of a non-numeric character will result in a data interrupt.

CHAPTER 3

THE BRIDGE FILES

File Description - Bridge File

The record format of the bridge file is:

Columns	Length	Format	Item	Data Element
1	1	char(1)		Delete byte
2-17	16	char(16)	1-1	Key
18	1	char(1)	1-3	Remark
19-34	16	char(16)	1-4	Coincident key
35-36	2	fixed(2)	1-5	Maintenance district
37-38	2	fixed(2)	1-6	Construction district
39-4Ø	2	fixed(3)	1-7	Detour length
41-65	25	char(25)	1-8	Features intersected
66-67	2	fixed(2)	1-9	Min. vertical clearance - feet
68-69	2	fixed(2)	1-9	Min. vertical clearance - inches
7Ø-71	2	fixed(3,1)	1-1Ø	Total horizontal clearance
72	1	char(1)	1-11	Major or minor
73-74	2	fixed(2)	2-1	Latitude - degrees
75-76	2	fixed(3,1)	2-1	Latitude - minutes
77-78	2	fixed(3)	2-2	Longitude - degrees
79-8Ø	2	fixed(3,1)	2-2	Longitude - minutes
81-85	5	fixed(9)	2-3	Inventory route
86-1ØØ	15	char(15)	2-4	Facility carried
1Ø1	1	fixed(1)	2-5	Physical vulnerability
1Ø2	1	fixed(1)	2-6	Custodian
1Ø3-1Ø4	2	fixed(2)	2-7	Year built
1Ø5-1Ø6	2	fixed(2)	2-8	Year improved
1Ø7	1	fixed(1)	2-9	Number of lanes on structure
1Ø8	1	fixed(1)	2-1Ø	Number of lanes under structure
1Ø9-111	3	fixed(4)	2-11	Design load
112	1	fixed(1)	2-12	Bridge median
113-114	2	fixed(2)	2-13	Skew
115	1	fixed(1)	2-14	Structure flared
116	1	fixed(1)	2-15	Navigation control
117-118	2	fixed(3)	2-16	Navigation vertical clearance
119-121	3	fixed(4)	2-17	Navigation horizontal clearance
122-123	2	fixed(2)	2-18	Type service
124-125	2	fixed(3)	3-1	Main structure type
126-127	2	fixed(3)	3-2	Approach structure type
128-129	2	fixed(3)	3-3	Number of spans in main unit
13Ø-132	3	fixed(4)	3-4	Number of approach spans
133-135	3	fixed(4)	3-5	Length of maximum span
136-139	4	fixed(6)	3-6	Structure length
14Ø-141	2	fixed(3,1)	3-7	Left sidewalk width
142-143	2	fixed(3,1)	3-8	Right sidewalk width
144-146	3	fixed(4,1)	3-9	Bridge roadway width
147-149	3	fixed(4,1)	3-1Ø	Bridge deck width
15Ø-153	4	fixed(6)	3-11	Station number
154-168	15	char(15)	3-12	Project number

Columns	Length	Format	Item	Data Element
169-170	2	fixed(2)	4-1	Min. vertical clearance - feet
171-172	2	fixed(2)	4-1	Min. vertical clearance - inches
173-174	2	fixed(2)	4-2	Min. vert. underclearance - feet
175-176	2	fixed(2)	4-2	Min. vert. underclearance - inches
177-178	2	fixed(3,1)	4-3	Min. lat. underclearance on right
179-180	2	fixed(3,1)	4-4	Min. lat. underclearance on left
181-183	3	fixed(4)	4-5	Wearing surface
184-185	2	fixed(2)	4-6	Wearing surface depth
186-189	4	char(4)	4-7	Bridge approach guardrail
190	1	char(1)	4-8a	Safety - Bridge rail
191	1	char(1)	4-8b	Safety - Transition
192	1	char(1)	4-8c	Safety - Approach guardrail
193	1	char(1)	4-8d	Safety - Guardrail terminal
194-195	2	fixed(2)	4-9	Posted speed limit
196-201	6	char(6)	4-10	Posted load limit
202	1	char(1)	4-11	Deck condition
203	1	char(1)	4-12	Superstructure
204	1	char(1)	4-13	Substructure
205	1	char(1)	4-14	Channel and channel protection
206	1	char(1)	4-15	Culvert and retaining walls
207-208	2	fixed(2)	4-16	Estimated remaining life
209-210	2	fixed(3)	4-17	Operating rating
211	1	char(1)	4-18	Approach roadway alignment
212-213	2	fixed(3)	4-19	Inventory rating
214	1	char(1)	4-20	Structural condition
215	1	char(1)	4-21	Deck geometry
216	1	char(1)	4-22	Underclearances, vert. and horiz.
217	1	char(1)	4-23	Safe load capacity
218	1	char(1)	4-24	Waterway adequacy
219	1	char(1)	4-25	Approach roadway alignment
220-221	2	fixed(2)	5-1	Year of needed improvement
222	1	fixed(1)	5-2	Type of service
223-224	2	fixed(3)	5-3	Type of work
225-228	4	fixed(6)	5-4	Length of improvement
229	1	fixed(1)	5-5	Proposed design loading
230-232	3	fixed(4)	5-6	Proposed road width
233-234	2	fixed(2)	5-7	Proposed number of lanes
235-238	4	fixed(6)	5-8	Design ADT
239-240	2	fixed(2)	5-9	Year of estimated ADT
241-242	2	fixed(2)	5-10	Year of roadway improvements
243	1	fixed(1)	5-11	Type of roadway improvements
244-246	3	fixed(5)	5-12	Cost of improvements
247-250	4	fixed(6)	5-13	Inspection date
251-256	6	char(6)	5-14	Structure batch serial number
257-262	6	char(6)	5-15	Microfilm serial number
263-264	2	fixed(2)		Date of update - month
265-266	2	fixed(2)		Date of update - day
267-268	2	fixed(2)		Date of update - year

The contents of these data elements are described in the publication Highway Information System Release 4.Ø - Data Coding Manual. The item numbers listed under the column "Item" are the item numbers used in that manual.

All data elements are stored in standard IBM character or packed decimal (fixed) formats. For character formats, the length specification (as in char(16)) gives the length in characters. For fixed formats, the first or only specification gives the total number of digits in the field and the second specification is the number of digits that lie to the right of an assumed decimal point.

When a record is deleted from the file, it is not physically removed until the file is reorganized. The delete byte in column one is used by the system to indicate deleted records.

The date of update field is added to the record automatically by the update program.

File Description - Bridge Report File

The record format of the bridge report file is:

Columns	Length	Format	Contents
1	1	fixed(1)	Signed route system
2-8	7	char(7)	Signed route number
9-13	5	char(5)	Defense section number
14-17	4	fixed(7,3)	Section length
18-21	4	fixed(7,3)	Miles from start of section
22-37	16	char(16)	Bridge key
38-39	2	char(2)	Remark
4Ø	1	char(1)	Bypassable
41-42	2	fixed(3)	City number
43-45	3	fixed(5)	ADT
46	1	fixed(1)	Design load
47-48	2	fixed(3)	Capacity
49-5Ø	2	fixed(2)	Vertical clearance - feet
51-52	2	fixed(2)	Vertical clearance - inches
53-54	2	fixed(3,1)	Horizontal clearance
55	1	fixed(1)	Number of lanes
56-59	4	fixed(6)	Total length
6Ø-62	3	fixed(4)	Span length
63-66	4	char(4)	Bridge type
67-68	2	fixed(2)	Year built
69-93	25	char(25)	Name of feature crossed
94-95	2	fixed(2)	County number
96	1	fixed(1)	Physical vulnerability
97-1ØØ	4		Latitude
1Ø1-1Ø4	4		Longitude

The latitude and longitude are broken down as follows:

Columns	Length	Format	Contents
97-98	2	fixed(2)	Latitude - degrees
99-1ØØ	2	fixed(3,1)	Latitude - minutes
1Ø1-1Ø2	2	fixed(3)	Longitude - degrees
1Ø3-1Ø4	2	fixed(3,1)	Longitude - minutes

The bridge report file is a sequential file rather than a keyed file. The records are sorted in order based on these fields:

1. Signed route system.
2. Defense section number.
3. Section length.
4. Miles from start of section.
5. Remark.

The signed route system field contains one of these values:

1 Interstate system
2 All other systems

The signed route number is derived from the inventory route (item 2-3) of the bridge file.

The defense section number comes from the defense cross-reference file. The section length is computed from the starting and ending keys of the defense section. The miles from start of section is computed from the starting key of the defense section and from the bridge key.

The remark field is given a value depending upon the bridge remark:

Report Remark	Bridge Remark
blank	Any but those shown in this table
P	P
T	T
U	U - Underpass under defense highway
UA	U - All other underpasses

The second character of the remark can also be filled in depending upon the type service field (item 2-18) of the bridge file:

Report Remark	Type service - first digit
x1	6
x2	7
x3	8

The bypassable code contains an N if the bridge detour length (item 1-7) is 2 or larger, and contains a blank otherwise.

The city number and county number are obtained from the roadlog file. The ADT is the recent-year ADT from the traffic file (interpolated if necessary).

The 1-digit design load is derived from the 4-digit design load of the bridge file. Details are given below in the section "The BDGDLOD Subroutine."

The following items are taken directly from the bridge file:

Bridge Item Number	Data Element
4-17	Capacity (operating rating)
1-9	Minimum vertical clearance
1-1Ø	Total horizontal clearance
2-9,1Ø	Number of lanes on/under structure
3-6	Total length
3-5	Span length
1-8	Features intersected
2-7	Year built
2-5	Physical vulnerability
2-1	Latitude
2-2	Longitude

The char(4) bridge type is calculated from the fixed(3) main structure type (item 3-1). Details are given below in the section "The BDGTYPE Subroutine."

The BDGRDQ Subroutine

BDGRDQ is a "low-level" subroutine for reading the bridge file sequentially. It provides the basic access functions, but does not incorporate higher-level functions such as the select capability implemented in the BDGRD subroutine.

BDGRDQ has these entry points:

BDGRDQF	Initialization - open the file
BDGRDQR	Set startkey-endkey range
BDGRDQ	Sequential read
BDGRDQX	Direct read - key-high
BDGRDQP	Position - key-high
BDGRDQC	Close the file
BDGRDQI	Combines functions of BDGRDQF and BDGRDQR

BDGRDQF Entry Point - Call BDGRDQF to open the file. Pass a binary fixed(15) value of 2.

BDGRDQR Entry Point - BDGRDQR can be called to limit access to a portion of the file based on key. Pass a char(16) starting key and a char(16) ending key. After the call, the program will have access to only those records having a key equal to or larger than the starting key and equal to or less than the ending key.

BDGRDQ Entry Point - Call BDGRDQ to read a record. Pass a pointer variable. A record is read and its address returned in the pointer. At end-of-file, a null value is returned.

BDGRDQX Entry Point - BDGRDQX can be called to read direct. Pass a char(16) key and a pointer. The first record in the file with a key equal to or larger than the passed key is read and its address is returned in the pointer. A null value is returned if the passed key is larger than that of the last record in the file.

BDGRDQP Entry Point - BDGRDQP can be called to position key-high without actually reading a record. Pass a char(16) key. A call to BDGRDQP followed by a call to BDGRDQ is equivalent to a call to BDGRDQX.

BDGRDQC Entry Point - Call BDGRDQC to close the file. No arguments are passed.

BDGRDQI Entry Point - A call to this entry point replaces a call to BDGRDQF followed by a call to BDGRDQR. Pass a binary fixed(15) value of 2, a char(16) starting key, and a char(16) ending key.

Error Messages - If the calling program fails to open the file or if it passes a value other than 2 when it opens the file, the following message is printed:

    ***** BDGRDQ-Ø1 - BDGRDQ NOT PROPERLY INITIALIZED *****

This error message is accompanied with a user-1ØØ abend.

The BDGRD Subroutine

BDGRD is a "high-level" subroutine for reading the bridge file. It automatically handles these parameters from HIS commands:

MAX-#-ENTRIES
DATA, START-MILEPOINT, and END-MILEPOINT
SELECT-DD and SELECT-SIZE

BDGRD has the following entry points:

BDGRDF	Initialization - open the file
BDGRD	Sequential read
BDGRDC	Close the file

BDGRDF Entry Point - Call BDGRDF to open the file. Pass a binary fixed(15) value of 2.

BDGRD Entry Point - Call BDGRD to read a record. Pass a pointer variable. A record is read and its address returned in the pointer. A null pointer is returned at end of file.

BDGRDC Entry Point - Call BDGRDC to close the file. No arguments are passed.

Error Messages -

***** BDGRD-Ø1 - MAX-#-ENTRIES COMPLETED *****

This message is informatory only, and indicates that processing was terminated because of a MAX-#-ENTRIES parameter.

***** BDGRD-Ø2 - NO DATA PARAMETER *****

This message is printed if the DATA parameter is omitted on the command.

The BDRRDQ Subroutine

BDRRDQ is a "low-level" subroutine for reading the bridge report file. It has the following entry points:

BDRRDQF	Initialization - open the file
BDRRDQ	Sequential read
BDRRDQC	Close the file

BDRRDQF Entry Point - Call BDRRDQF to open the file. Pass a binary fixed(15) value of 2.

BDRRDQ Entry Point - Call BDRRDQ to read a record. Pass a pointer variable. A record is read and its address is returned in the pointer. At end of file, a null pointer is returned.

BDRRDQC Entry Point - Call BDRRDQC to close the file. No arguments are passed.

The BDGIRTE Subroutine

BDGIRTE breaks the fixed(9) inventory route code into its five components. Pass these arguments:

1. Inventory route - fixed(9)
2. Carried by/under - fixed(1)
3. Highway type - fixed(1)
4. Additional designation - fixed(1)
5. Route number - fixed(5)
6. Direction - fixed(1)

The BDGTYPE Subroutine

BDGTYPE derives a char(4) description from either the main structure type or the approach structure type code. Pass the fixed(3) type code and a char(4) variable. The char(4) variable must be passed without the standard PL/I dope vector - specify OPTIONS (ASM INTER) on the entry point declaration, or overlay a fixed(7) variable on the char(4) variable and pass the fixed(7) variable. A list of codes returned follows:

Type Code	Description
1Ø1,2Ø1,5Ø1,6Ø1	SLAB
1Ø4,2Ø4	TBM
1Ø5,1Ø6,2Ø5,2Ø6,5Ø5,5Ø6,6Ø5,6Ø6	CBG
5Ø2,6Ø2	P/CB
1Ø7,2Ø7	CF
3Ø2,4Ø2	SMS
3Ø3,3Ø8,4Ø3,4Ø8	SG
3Ø5,3Ø6,4Ø5,4Ø6	SBG
3Ø9,4Ø9	STD
31Ø,41Ø	STT
3Ø7,4Ø7	SF
311,312,411,412	SA
111,112,211,212,811	MA
7Ø9,71Ø,711,712	TA
7ØØ,7Ø2,7Ø3	TS
313,414	SUS
315,316,317	MOV
Ø18,118,318,518,718,818,918	TUN
Ø19,119,219,319,519,719,819,919	CULV
ØØØ	blank
other	OTH

The BDGSRTE Subroutine

BDGSRTE derives the char(7) signed route number from the fixed(9) inventory route code. Pass the fixed(9) inventory route and a char(7) variable for the signed route number. The char(7) variable must be passed without a standard PL/I dope vector - either specify OPTIONS (ASM INTER) in the entry point declaration or declare a fixed(15) variable overlaid on the char(7) variable and pass the fixed(15) variable.

The BDGDLOD Subroutine

BDGDLOD derives the FHWA 1-digit design load code from the 4-digit design load code used in the bridge file. Pass the fixed(4) design load and a fixed(1) variable for the FHWA code. The code is derived as follows:

FHWA Code	Design Load
7	ØØØ7
8	ØØØ8
9	ØØØ9
Ø	ØØØØ
1	ØØØ1-1499 (except ØØØ7,ØØØ8,ØØØ9)
2	15xx-19xx (xx non-zero)
3	15xx-19xx (xx zero)
4	2Øxx-99xx (xx non-zero)
5	2Øxx-24xx (xx zero)
6	25xx-99xx (xx zero)

The BDGCVT Subroutine

BDGCVT can convert bridge "data strings" into record format and vice versa. Data strings are concatenations of bridge data cards in this format:

Columns	Contents
1	End-of-file code
2-17	Key
18-74	Columns 18-74 of card 1
75-135	Columns 18-78 of card 2
136-193	Columns 18-75 of card 3
194-249	Columns 18-73 of card 4
25Ø-3Ø2	Columns 18-7Ø of card 5

BDGCVT Entry Point - Call BDGCVT to convert a data string into record format. Pass the char(268) bridge record, the char(3Ø2) data string, and a binary fixed(15) return code. The return code is set to one of these codes:

Ø Successful
1 Unsuccessful - Non-numeric character in numeric field

BDGCVTA Entry Point - Call BDGCVTA to convert a record into data string format. Pass the char(268) bridge record and the char(3Ø2) data string.

Error Messages - If a return code of 1 is returned to BDGCVT entry point, this message is printed:

***** ERROR - A NON-NUMERIC CHARACTER WAS DETECTED IN A NUMERIC FIELD

The BDGINB Subroutine

BDGINB is a subroutine for updating the bridge file. It has these entry points:

BDGINB	Add a record to the file
BDGINBC	Close the file

BDGINB Entry Point - Call BDGINB to add a record to the file. Pass a pointer that contains the record's address and a binary fixed(15) return code. Return codes are:

Ø	Successful
1	Unsuccessful - a record already existed with the new record's key

BDGINBC Entry Point - Call BDGINBC to close the file. No arguments are passed.

The BDGRWB Subroutine

BDGRWB is a subroutine for updating the bridge file. It has these entry points:

BDGRWB	Read a record for subsequent rewrite or delete
BDGRWBR	Rewrite a record
BDGRWBD	Delete a record
BDGRWBC	Close the file

BDGRWB Entry Point - Call BDGRWB to read a record. Pass a pointer variable and a char(15) key. The record's address is returned in the pointer. If the record does not exist, a null pointer is returned.

BDGRWBR and BDGRWBD Entry Points - Call one of these entry points after reading a record. Pass a binary fixed(15) return code. Return codes are:

Ø	Successful
1	Unsuccessful

BDGRWBC Entry Point - Call BDGRWBC to close the file. No arguments are passed.

CHAPTER 4

THE RAILROAD FILES

File Description - Railroad File

Railroad records are stored in the following format:

Columns	Length	Format	Item	Data Element
1	1	char(1)		Delete byte
2-16	15	char(15)	a-2	Key
17-18	2	fixed(2)	a-3	Survey month
19-20	2	fixed(2)	a-3	Survey day
21-22	2	fixed(2)	a-3	Survey year
23	1	char(1)	b-21	Urban/rural
24-48	25	char(25)	b-22	Location
49-50	2	fixed(2)	a-4	Road width
				(ROAD APPROACH ONE)
51	1	fixed(1)	a-5	Direction
52-53	2	fixed(3)	a-6	Angle at O
54-56	3	fixed(4)	a-7	Sight distance left
57-59	3	fixed(4)	a-8	Sight distance right
60-61	2	fixed(3)	a-9	Distance OP
62-63	2	fixed(2)	a-10	Curvature degrees
64-65	2	fixed(2)	a-11	Curvature minutes
66-67	2	fixed(2)	a-12	Grade of road
68	1	fixed(1)	a-13	Local interference
69	1	fixed(1)	a-14	Vertical sight restriction
				(ROAD APPROACH TWO)
70	1	fixed(1)	a-15	Direction
71-72	2	fixed(3)	a-16	Angle at O
73-75	3	fixed(4)	a-17	Sight distance left
76-78	3	fixed(4)	a-18	Sight distance right
79-80	2	fixed(3)	a-19	Distance ON
81-82	2	fixed(2)	a-20	Curvature degrees
83-84	2	fixed(2)	a-21	Curvature minutes
85-86	2	fixed(2)	a-22	Grade of road
87	1	fixed(1)	a-23	Local interference
88	1	fixed(1)	a-24	Vertical sight restriction
89	1	fixed(1)	b-3	Daily passenger trains
90-91	2	fixed(2)	b-4	Daily passenger speed
92-93	2	fixed(2)	b-5	Daily freight trains
94-95	2	fixed(2)	b-6	Daily freight speed
96-97	2	fixed(3)	b-7	Daily switch trains
98-99	2	fixed(2)	b-8	Daily switch speed
100	1	fixed(1)	b-9	Weekly passenger trains
101-102	2	fixed(2)	b-10	Weekly passenger speed
103-104	2	fixed(2)	b-11	Weekly freight trains
105-106	2	fixed(2)	b-12	Weekly freight speed

(continued on next page)

Columns	Length	Format	Item	Data Element
1Ø7-1Ø8	2	fixed(3)	b-13	Weekly switch trains
1Ø9-11Ø	2	fixed(2)	b-14	Weekly switch speed
111	1	fixed(1)	b-15	Seasonal passenger trains
112-113	2	fixed(2)	b-16	Seasonal passenger speed
114-115	2	fixed(2)	b-17	Seasonal freight trains
116-117	2	fixed(2)	b-18	Seasonal freight speed
118-119	2	fixed(3)	b-19	Seasonal switch trains
12Ø-121	2	fixed(2)	b-2Ø	Seasonal switch speed
122-123	2	fixed(2)	c-3	Inventory month
124-125	2	fixed(2)	c-3	Inventory day
126-127	2	fixed(2)	c-3	Inventory year
128-131	4	char(4)	c-4	Operating RR
132-138	7	char(7)	c-5	Crossing ID
139-153	15	char(15)	c-6	Branch/line
154-157	4	fixed(6,2)	c-7	Branch/line milepost
158	1	fixed(1)	c-8	Main tracks
159-16Ø	2	fixed(2)	c-9	Other tracks
161	1	fixed(1)	c-1Ø	Other railroad use code
162-165	4	char(4)	c-11	Other railroad name
166	1	fixed(1)	c-12	Number of crossbucks
167	1	fixed(1)	c-13	Number of stop signs
168	1	fixed(1)	c-14	Number of wigwags
169	1	fixed(1)	c-15	Number of flashing lights
17Ø	1	fixed(1)	c-16	Number of gates
171	1	fixed(1)	c-17	Number of lanes
172-173	2	fixed(3)	c-18	Number of nightly trains
174-176	3	fixed(4)	c-19	Estimated ADT
177-179	3			Unused at this time
18Ø-181	2	fixed(2)		Month of update
182-183	2	fixed(2)		Day of update
184-185	2	fixed(2)		Year of update

The contents of these data elements are described in the publication Highway Information System Release 4.Ø - Data Coding Manual. The item numbers listed under the column "Item" are the item numbers used in that manual.

All data elements are stored in standard IBM character or packed decimal (fixed) formats. For character formats, the length specification given above is the length in characters. For fixed formats, the first or only length specification is the length in digits. The second specification is the number of those digits that are treated as being to the right of an assumed decimal point.

When a record is deleted from the file, it is not physically removed until the file is reorganized. The delete byte in column one is used by the system to indicate deleted records. Any records containing the delete

code (X'FF') in this position are ignored by all programs.

The month, day and year of update fields are automatically handled by the update program.

File Description - Railroad Report File

Railroad report records are stored in the following format:

Columns	Length	Format	Contents
1	1	char(1)	Delete byte
2-16	15	char(15)	Key
17-41	25	char(25)	Verbal description of location
42-56	15	char(15)	County
57	1	char(1)	Urban/rural code
58-6∅	3	fixed(5)	ADT
61-64	4	char(4)	Operating RR Co.
65	1	fixed(1)	Daily passenger trains
66-67	2	fixed(2)	Daily freight trains
68-69	2	fixed(3)	Daily switch trains
7∅	1	fixed(1)	Weekly passenger trains
71-72	2	fixed(2)	Weekly freight trains
73-74	2	fixed(3)	Weekly switch trains
75	1	fixed(1)	Seasonal passenger trains
76-77	2	fixed(2)	Seasonal freight trains
78-79	2	fixed(3)	Seasonal switch trains
8∅-82	3	fixed(5,1)	Hazard index
83-91	9	char(9)	Type of protection
92-94	3	fixed(5,1)	Adjusted hazard index
95	1	char(1)	ADT flag

When the ADT flag is an '*' the ADT field is either the average ADT over three years (ADT subroutine, page 8-7), or is the estimated ADT from the railroad file.

The RRXRDQ Subroutine

RRXRDQ is a "low-level" subroutine for reading the railroad file. It provides the basic access functions but not the higher-level functions such as the select capability implemented in the RRXRD subroutine.

RRXRDQ has the following entry points:

RRXRDQF	Initialization - open the file
RRXRDQR	Set startkey-endkey range
RRXRDQ	Sequential read
RRXRDQX	Direct read - key high
RRXRDQP	Position - key high
RRXRDQC	Close the file
RRXRDQI	Combines functions of RRXRDQF and RRXRDQR

RRXRDQF Entry Point - Call RRXRDQF to open the file. Pass a binary fixed(15) value of 2.

RRXRDQR Entry Point - RRXRDQR can be called to limit access to a portion of the file. Pass a char(15) starting key and a char(15) ending key. After the call, access is limited to those records having a key equal to or larger than the starting key and equal to or smaller than the ending key.

RRXRDQ Entry Point - Call RRXRDQ to read a record. Pass a pointer variable. A record is read and its address is returned in the pointer. At end of file, a null value is returned.

RRXRDQX Entry Point - RRXRDQX performs a direct read operation. Pass a char(15) key and a pointer variable. The first record in the file with a key equal to or larger than the passed key is read and its address is returned in the pointer. A null pointer is returned if the passed key is larger than that of the last record in the file.

RRXRDQP Entry Point - RRXRDQP performs a key-high search without actually reading a record. Pass a char(15) key. A RRXRDQP call followed by a RRXRDQ call is equivalent to a RRXRDQX call.

RRXRDQC Entry Point - Call RRXRDQC to close the file. No arguments are passed.

RRXRDQI Entry Point - A call to RRXRDQI is equivalent to a call to RRXRDQF followed by a call to RRXRDQR. Pass a binary fixed(15) value of 2, a char(15) starting key, and a char(15) ending key.

The RRXRD Subroutine

RRXRD is a "high-level" subroutine for reading the railroad file. It should be used by programs that print listings or summaries from the railroad file so that the select capabilities will be implemented. Only one high-level subroutine can be called by any one program.

RRXRD automatically handles the following parameters from HIS commands without intervention from the calling program:

DATA, START-MILEPOINT, and END-MILEPOINT
SELECT-DD and SELECT-SIZE
MAX-#-ENTRIES

A DATA parameter must be coded on the command. The remaining parameters are optional.

RRXRD has the following entry points:

RRXRDF	Initialization - open the file
RRXRD	Sequential read
RRXRDC	Close the file

RRXRDF Entry Point - Call RRXRDF to open the file. Pass a binary fixed(15) value of 2.

RRXRD Entry Point - Call RRXRD to read a record. Pass a pointer variable. A record is read and returned in the pointer. At end of file, a null pointer is returned.

RRXRDC Entry Point - call RRXRDC to close the file. No arguments are passed.

Error Messages -

***** RRXRD-Ø1 -- MAX-#-ENTRIES COMPLETED *****

This message is informatory only, and indicates that processing was terminated because of a MAX-#-ENTRIES parameter.

***** RRXRD-Ø2 -- NO DATA PARAMETER *****

The DATA parameter was omitted from the command. A user-1ØØ abend accompanies this message.

The RRXINB Subroutine

RRXINB is a subroutine for updating the railroad file. It can add records to the file and perform new-key operations. It has the following

entry points:

RRXINB	Add a record to the file
RRXINBN	Change a record's key
RRXINBC	Close the file

Calling sequences and results are the same as for RLGINB (see page 5-9).

Error Messages -

***** I/O ERROR *****

An input/output error was detected and a user-1ØØ abend follows.

***** ATTEMPT TO INSERT OVER EXISTING RECORD *****

The key of this record already exists in the file.

***** NEW-KEY RECORD ALREADY EXISTS IN FILE *****

This is the same as the above message only comes from the RRXINBN entry point.

***** OLD-KEY RECORD DOES NOT EXIST IN FILE *****

The key to be changed could not be found in the file.

The RRXRWB Subroutine

RRXRWB is a subroutine for updating the railroad file. It can rewrite and delete records. It has these entry points:

RRXRWB	Read a record for subsequent rewrite or delete
RRXRWBR	Rewrite a record
RRXRWBD	Delete a record
RRXRWBC	Close the file

Calling sequences are the same as for RLGRWB (see page 5-1Ø).

Error Messages -

***** PROGRAM ERROR: INVALID FILE ATTRIBUTES *****

The file DCB is inconsistant with attributes expected by this subroutine. A user-1ØØ abend follows.

***** I/O ERROR *****

An input/output error was detected and a user-1ØØ abend follows.

***** PROGRAM ERROR: RWBD OR RWBR INVOKED WITHOUT PREVIOUS READ *****

The program called RRXRWBR or RRXRWBD without a previous successful call to RRXRWB, or the program changed the key of a record. A user-1ØØ abend follows.

The RRRWRQ Subroutine

RRRWRQ is a subroutine for loading the railroad report file. It has these entry points:

RRRWRQ	Write a record
RRRWRQC	Close the file

RRRWRQ Entry Point - Call RRRWRQ to write a record. Pass a pointer that contains the address of the record. Records must be written in order by key.

RRRWRQC Entry Point - Call RRRWRQC to close the file. No arguments are passed.

The RRXCVT Subroutine

RRXCVT can convert railroad "data strings" into railroad record format and vice versa. A data string is a concatenation of railroad data cards used by the update program, and has this format:

Columns	Length	Format	Contents
1	1	char(1)	Unused
2-16	15	char(15)	Key
17-72	56	char(56)	Columns 17-72 of the A card
73-134	62	char(62)	Columns 17-78 of the B card
135-194	6Ø	char(6Ø)	Columns 17-76 of the C card

RRXCVT has these entry points:

RRXCVT	Convert a data string to record format
RRXCVTA	Convert a record to data string format

RRXCVT Entry Point - Pass a pointer that has the address of a 176-character area for storing the record, a pointer that contains the data string address, and a binary fixed(15) return code argument.

Return codes are:

∅	Successful
1	Unsuccessful - non-numerics in a numeric field

A return code of 1 is accompanied by an error message.

RRXCVTA Entry Point - Pass a pointer that contains the record address, and a pointer that contains the address of a 194-character area for storing the data string.

Error Messages - If a return code of 1 is returned by the RRXCVT entry point, this message is printed:

***** (E) CONVERSION ERROR - NON-NUMERIC CHARACTER IN NUMERIC FIELD

CHAPTER 5

THE ROADLOG FILE

File Description

The record format of roadlog records is:

Columns	Length	Format	Item	Data Element
1	1	char(1)		Delete byte
2-16	15	char(15)	A-1	Key
17-18	2	char(2)	A-5	Remark
19-21	3	fixed(5,3)	B-17	Section length
22-24	3	fixed(5,3)	B-13	Route length
25-27	3	fixed(5,3)	B-14	Constructed length
28-30	3	fixed(5,3)	B-15	Unimproved length
31-32	2	fixed(3,3)	B-16	Wye length
33-67	35	char(35)	A-2	Description
68-78	11	char(11)	A-4	Project number
79	1	char(1)	B-3	Divided/undivided code
80	1	fixed(1)	B-2	Number of lanes
81	1	fixed(1)	B-19	One-way code
82-83	2	fixed(3)	B-5	City number
84-85	2	fixed(2)	A-6	County number
86-87	2	fixed(2)	B-6	Year built
88-89	2	fixed(2)	B-7	Year improved
90-91	2	fixed(2)	A-7	Forest highway number
92-93	2	fixed(2)	C-3	Jurisdiction
94-95	2	fixed(2)	A-9	Location code 1
96-97	2	fixed(2)	A-9	Location code 2
98-99	2	fixed(2)	B-22	Maintenance division
100-101	2	fixed(2)	B-11	Surface width
102-103	2	fixed(2)	B-12	Roadway width
104-105	2	fixed(2,1)	B-9	Surface thickness
106-107	2	fixed(3,1)	B-10	Base thickness
108	1	fixed(1)	A-11	Control of access
109-111	3	fixed(4)	B-8	Surface type
112-114	3	fixed(4)	A-3	Maintenance section
115-116	2	fixed(2)	B-18	Effective date - month
117-118	2	fixed(2)	B-18	Effective date - day
119-120	2	fixed(2)	B-18	Effective date - year
121	1	fixed(1)	A-10	Functional classification
122-123	2	fixed(2)	B-20	Construction division
124-125	2	fixed(2)	B-21	Planning division
126-127	2	fixed(2)	C-4	Area name
128-129	2	fixed(2)	C-5	Map sheet
130-137	8	char(8)	C-2	Range, township, and section
138-139	2	fixed(2)		Date of update - month
140-141	2	fixed(2)		Date of update - day
142-143	2	fixed(2)		Date of update - year
144-148	5	char(5)	B-23	Defense section number
149-160	12			Unused at this time

The contents of these data elements are described in the publication Highway Information System Release 4.Ø - Data Coding Manual. The item numbers listed under the column "Item" are the item numbers used in that manual.

All data elements are stored in standard IBM character or packed decimal (fixed) formats. For character formats, the length specification (as in char(15)) gives the length in characters. For packed decimal formats with one length specification (as in fixed(2)), the length is the number of decimal digits. For packed decimal formats with two length specifications (as in fixed(5,3)), the first length is the total number of digits in the field and the second length is the number of digits that are assumed to lie to the right of an assumed decimal point.

When a record is deleted from the file, it is not physically removed until the file is reorganized. The delete byte in column one is used by the system to indicate deleted records. Any records containing the delete code (X'FF') in this position are ignored by all programs.

The date of update field is added to the record by the update program. This function is automatic and requires no coding.

The admistration code (item A-8) and the population code (item B-4) coded on roadlog data cards are not stored in the records. The administration code can be derived from the project number. The population code can be derived from the city number. The following subroutines, described in the chapter entitled "HIS.TABLES" in this manual, can be used to derive these codes:

INPROJ	Administration code
CVTPROJ	Administration code
INCITY	Population code
GETCITY	Population code

Confusion sometimes arises over the terminology "municipal mileage" and "urban mileage." Municipal mileage is mileage located within the city limits of an incorporated city. Any record that contains a value of Ø1 in the first location field is municipal mileage. Urban mileage is mileage located with the urban limits of an urban area. Urban mileage consists of "rural urban" mileage (located within the urban area but outside the city limits) and "municipal urban" mileage (located within both the urban area and the city limits). Rural urban mileage is stored with a value of Ø2 in the first location field. Municipal urban is stored with a value of Ø1 in the first location field and a population code (derived from the city number) of 4 or larger.

The RLGRDQ Subroutine

RLGRDQ is a "low-level" subroutine for reading the roadlog file. It provides the basic access functions, but does not incorporate higher-level functions such as the select capability implemented in the RLGRD subroutine.

RLGRDQ has the following entry points:

RLGRDQF	Initialization - open the file
RLGRDQR	Set startkey-endkey range
RLGRDQ	Sequential read
RLGRDQX	Direct read - key-high
RLGRDQL	Direct read - key-low
RLGRDQM	Direct read - key-low and key-equal
RLGRDQP	Position - key-high
RLGRDQT	Set record type
RLGRDQC	Close the file
RLGRDQ1	Save current environment
RLGRDQ2	Restore previous environment
RLGRDQI	Combines functions of RLGRDQF and RLGRDQR

RLGRDQF Entry Point - Before the calling program can request data from the roadlog file, the subroutine must be initialized. Initialization is accomplished by calling RLGRDQF. Pass a binary fixed (15) value of 2, as in this example:

```
DECLARE RLGRDQF ENTRY (BIN FIXED);
CALL RLGRDQF (2);
```

Initialization can be accomplished by calling RLGRDQI instead of RLGRDQF - see below.

RLGRDQR Entry Point - The calling program can limit access to a portion of the file by calling RLGRDQR. Pass a starting key and an ending key (each key is char(15)). After the call, the program will have access to only those records with a key larger than or equal to the starting key and smaller than or equal to the ending key.

RLGRDQ Entry Point - A call to this entry point returns one record from the file. Records are read sequentially in order by key. Pass a pointer variable. The subroutine will read a record and store its address in the pointer. At end of file, a NULL value (X'FFØØØØØØ') is stored in the pointer. A call to

RLGRDQ is roughly equivalent to the PL/I statements:

```
ON ENDFILE (ROADLOG) PTR=NULL();
READ FILE (ROADLOG) SET (PTR);
```

RLGRDQX Entry Point - RLGRDQX allows access to the file by key. Pass a char(15) key and a pointer. The subroutine locates the first record in the file that has a key equal to or larger than the passed key and returns its address in the pointer. A NULL value is returned if the passed key is larger than that of any record in the file. A RLGRDQX is roughly equivalent to the PL/I statement:

```
READ FILE (ROADLOG) SET (PTR) KEY (KEY);
```

It is not exactly equivalent because PL/I does not implement a key-high function. PL/I has a key-equal function in which the key specified must be exactly equal to a key in the file and a genkey function in which a portion of the key is specified and the first record with a key starting with the specified portion is retrieved.

RLGRDQL Entry Point - RLGRDQL allows access to the file via a key-low function. IBM does not have a key-low function implemented, and use of this entry point may cause substantial overhead due to the search that is required. The calling sequence is the same as for RLGRDQX - pass a char(15) key and a pointer. A null value will be returned if any of the following conditions occur:

1. No records are stored having the route number specified in columns 1-6 of the passed key.
2. The passed key is beyond the end of the route.

The record returned will always be a mileage record or a CO record.

RLGRDQM Entry Point - RLGRDQM is similar to RLGRDQL, except that two pointers are passed (pass a char(15) key and two pointers). A null value is returned in both pointers if either of the conditions in RLGRDQL above occur. The first pointer is used for the key-low record and the second is used for the key-equal record. If the key of the first record of a route is passed, the key-low pointer will be set to null and the key-equal

pointer will be set to the address of the first record of the route. If the key of a record in the middle of the route is passed, the key-equal pointer will be set to its address and the key-low pointer will be set to the address of the preceding record. If a key between two records is passed, the key-low pointer will be set to the address of the record whose key is smaller than the passed key and the key-equal pointer will be set to null. Only mileage records and CO records are returned.

RLGRDQP Entry Point - This entry point can be used to position key-high to a record in the file. The record is not read until a subsequent RLGRDQ call. Pass a char(15) key. A RLGRDQP call followed by a RLGRDQ call is identical to a RLGRDQX call.

RLGRDQT Entry Point - RLGRDQT provides the capability of excluding records based on the contents of the remarks field. Pass a binary fixed (15) value computed by summing the desired types:

1	Mileage records - codes blank, OS, NE, SP, and LP
2	CO records
4	EN records
8	IL records
16	DS and ER records
32	All other records

For example, to read only mileage records, CO records, and EN records, pass a value of 7 (1 + 2 + 4). This function is ignored when RLGRDQL and RLGRDQM are called.

RLGRDQC Entry Point - When the program has finished accessing the roadlog file, it must call RLGRDQC to close the file. No arguments are passed. A system-CØ3 abend may result if the file is not closed.

RLGRDQ1 and RLGRDQ2 Entry Points - These entry points are implemented for use by the select routines. When the select routines access the roadlog file, they first call RLGRDQ1 to save the environment of the mainline program and they call RLGRDQ2 to restore this environment when they have finished accessing the file.

RLGRDQI Entry Point - This entry point can be used to combine the functions of RLGRDQF and RLGRDQR. Pass a binary fixed (15) value of 1, a char(15) starting key, and a char(15) ending key.

Error Messages - When an error is detected, the subroutine will print an appropriate message. The messages implemented at this time are:

***** RLGRDQ-Ø1 - RLGRDQ NOT PROPERLY INITIALIZED - ENTRY POINT name

The calling program either did not call RLGRDQF (or RLGRDQI), or passed a value other than 2.

***** RLGRDQ-Ø2 - INVALID RLGRDQ1 CALL

RLGRDQ1 was called twice in succession without an intervening call to RLGRDQ2.

***** RLGRDQ-Ø3 - INVALID RLGRDQ2 CALL

RLGRDQ2 was called but was not preceded by a RLGRDQ1 call.

Each of these messages is accompanied with a user-1ØØ abend.

Sample Program - This sample PL/I program illustrates the use of RLGRDQ to read the roadlog file. Only mileage records, CO records, and EN records are read.

```
/* SAMPLE PROGRAM - USE OF RLGRDQ */

SAMPLE:  PROCEDURE OPTIONS (MAIN);

DECLARE RLGPTR POINTER,
        RLGRDQ  ENTRY (POINTER),
        RLGRDQF ENTRY (BIN FIXED),
        RLGRDQT ENTRY (BIN FIXED),
        RLGRDQC ENTRY;

        CALL RLGRDQF (2);   /* INITIALIZE */
        CALL RLGRDQT (7);   /* SET RECORD TYPES */

LOOP:   CALL RLGRDQ (RLGPTR);    /* READ A RECORD */
        IF RLGPTR=NULL() THEN GOTO DONE;   /* TEST FOR ENDFILE */
              .
              .
              .
        GOTO LOOP;

DONE:   CALL RLGRDQC;   /* CLOSE THE FILE */
        END SAMPLE;
```

The RLGRD Subroutine

RLGRD is a "high-level" subroutine for reading the roadlog file. It should be used by programs that print listings or summaries of roadlog data so that the select capabilities will be implemented. It should not be used by programs that use the roadlog file as a subsidiary file. Only one high-level subroutine can be called by any one program.

RLGRD automatically handles the following parameters from HIS commands without intervention from the calling program:

DATA, START-MILEPOINT, and END-MILEPOINT
RLG-TYPES
SELECT-DD and SELECT-SIZE
MAX-#-ENTRIES

A DATA parameter must be coded on the command when RLGRD is used. The other parameters are optional.

RLGRD has the following entry points:

RLGRDF	Initialization - open the file
RLGRDT	Set record types
RLGRD	Sequential read
RLGRDX	Direct read - key-high
RLGRDC	Close the file

RLGRDF Entry Point - The calling program must call this entry point first. Pass a binary fixed(15) value of 2.

RLGRDT Entry Point - This entry point is used to select records based on the contents of the remarks field. The calling sequence and value passed are identical to those shown above for the RLGRDQT entry point of RLGRDQ.

RLGRD Entry Point - This entry point is called to read a record. Pass a pointer variable. Upon return, the address of the record is stored in the pointer. At end of file, a null pointer is returned.

RLGRDX Entry Point - This entry point allows access to the file based on key. Its use is identical to entry point RLGRDQX of RLGRDQ.

RLGRDC Entry Point - Upon completion, the calling program must call RLGRDC to close the file. No arguments are passed.

Error Messages - When an error is detected, the subroutine prints an appropriate message. The messages implemented are:

***** RLGRD-Ø1 - MAX-#-ENTRIES COMPLETED

This message is informatory only, and indicates that an endfile return was performed because the specified number of records had been processed.

***** RLGRD-Ø2 - NO DATA PARAMETER

No DATA parameter was coded on the command. A user-1ØØ abend is issued.

Sample Program - This sample PL/I program illustrates the use of RLGRD. Only mileage records, CO records, and EN records are read.

```
/* SAMPLE PROGRAM - USE OF RLGRD */
SAMPLE:  PROCEDURE OPTIONS (MAIN);
DECLARE RLGPTR POINTER,
        RLGRD  ENTRY (POINTER),
        RLGRDF ENTRY (BIN FIXED),
        RLGRDT ENTRY (BIN FIXED),
        RLGRDC ENTRY;
        CALL RLGRDF (2);     /* INITIALIZE */
        CALL RLGRDT (7);     /* SET RECORD TYPES */
LOOP:   CALL RLGRD (RLGPTR);    /* READ A RECORD */
        IF RLGPTR=NULL() THEN GOTO DONE;    /* TEST FOR ENDFILE */
             .
             .
             .
        GOTO LOOP;
DONE:   CALL RLGRDC;    /* CLOSE THE FILE */
        END SAMPLE;
```

When this program is executed, a DATA parameter must be specified on the command.

The RLGINB Subroutine

RLGINB is a subroutine for updating the roadlog file. It can be used to add records to the file and to perform new-key functions.

RLGINB has these entry points:

RLGINB	Add a record to the file
RLGINBN	Change a record's key
RLGINBC	Close the file

RLGINB Entry Point - Call RLGINB to add a record to the file. Pass either the char(16Ø) record or a pointer which has the address of the record and a binary fixed(15) return code variable. The return code will be set as follows:

Ø	Successful
1	Unsuccessful - a record already exists with the new record's key

RLGINB stores the current date in the date of update field of the record before writing the record to the file.

RLGINBN Entry Point - Call RLGINBN to modify the key of an existing record. Pass the char(15) key as currently stored, the char(15) replacement key, and a binary fixed(15) return code variable. The return code will be set as follows:

Ø	Successful
1	Unsuccessful - a record already exists with the replacement key
2	Unsuccessful - no record exists with current key specified

If the operation is successful, RLGINBN stored the current date in the date of update field of the record.

RLGINBC Entry Point - Call RLGINBC upon completion to close the file. No arguments are passed.

Error Messages - If an input/output error occurs, the subroutine prints the message ***** I/O ERROR ***** and issues a user-1ØØ abend.

The RLGRWB Subroutine

RLGRWB is a subroutine for updating the file. It can rewrite and delete records.

RLGRWB has these entry points:

RLGRWB	Read a record for subsequent rewrite or delete
RLGRWBR	Rewrite a record
RLGRWBD	Delete a record
RLGRWBC	Close the file

RLGRWB Entry Point - Call this entry point to read a record for rewriting or deleting. Pass a pointer and a char(15) key. If the record is found, its address is stored in the pointer. If the record is not found, a null value is stored in the pointer. The program can modify any part of the record except the key field.

RLGRWBR Entry Point - After reading a record and making the necessary modifications, call RLGRWBR to rewrite the record. The key field must not have been modified. No arguments are passed to RLGRWBR. RLGRWBR stores the current date in the date of update before rewriting the record.

RLGRWBD Entry Point - After reading a record, call RLGRWBD to delete the record. The key field must not have been modified. No arguments are passed to RLGRWBD.

RLGRWBC Entry Point - Call RLGRWBC upon completion to close the file. No arguments are passed.

Error Messages - If RLGRWB detects an error, it prints an error message and issues a user-1ØØ abend. Implemented messages are:

***** I/O ERROR *****

An input/output error was detected.

***** PROGRAM ERROR: RWBD OR RWBR INVOKED WITHOUT PREVIOUS READ *****

The program called RLGRWBR or RLGRWBD without a previous successful call to RLGRWB, or the program changed the key of a record.

The COINKEY Subroutine

COINKEY derives the starting and ending key from a CO or IL description. Pass the char(35) description, a char(15) variable for the starting key, and a char(15) variable for the ending key.

The KEYRLG Subroutine

KEYRLG checks a key against the roadlog file to determine whether the key lies within a legitimate section of road. Pass a char(15) key, a binary fixed (15) return code variable, and two additional char(15) variables. The return code and the two additional keys will be set as follows:

- Ø Key is valid - key1 and key2 are set to blanks.
- 1 Key is located within a PTW section - key1 and key2 are set to the beginning and ending keys of the PTW section
- 2 Key is located within a coincident section - key1 and key2 are set to the beginning and ending keys of the coincident section
- 3 Key is located beyond the end of the route - key1 is set to the key at the end of the route and key2 is filled with blanks
- 4 Route does not exist - key1 and key2 are filled with blanks

KEYRLG examines only federal aid routes - a return code of 4 will result from a call with a local system key.

Error Messages - When KEYRLG detects an error, it prints an appropriate message. Implemented messages are:

***** KEYRLG-Ø1 - ERROR IN ROADLOG FILE - ROUTE xxxxxx HAS NO EN RECORD

The route shown in the message has no EN route, so KEYRLG cannot determine the location of the end of the route. Execution continues, and the end is assumed to be at 999+9.999.

***** KEYRLG-Ø2 - TOO MANY ROADLOG ROUTES FOR PRESENT PROGRAM CAPABILITIES

***** KEYRLG-Ø3 - TOO MANY ROADLOG CO RECORDS FOR PRESENT PROGRAM CAPABILITIES

Either of these messages indicates that insufficient storage is available for storing roadlog records. The subroutine must be modified to increase storage. Either of these messages is accompanied with a user-1ØØ abend.

The RLGCVT Subroutine

RLGCVT can convert roadlog "data strings" into roadlog records and vice versa. A data string is a concatenation of roadlog data cards:

Columns	Contents
1	Blank
2-16	Key
17-80	Columns 17-80 of the A card
81-144	Columns 17-80 of the B card
145-200	Columns 17-72 of the C card

RLGCVT has these entry points:

RLGCVT	Convert a data string to record format
RLGCVTA	Convert a record to data string format

RLGCVT Entry Point - Call this entry point to convert a data string to record format. Pass a pointer that contains the address of a 160-character area into which the record will be placed and a pointer that contains the address of the 200-character data string. The program must have previously checked for non-numeric characters in fields that must be converted to packed decimal format - a non-numeric character in one of these fields will cause a data interrupt.

RLGCVTA Entry Point - Call this entry point to convert a record to data string format. Pass a pointer that contains the address of the 160-character record and a pointer that contains the address of a 200-character area into which the data string will be placed.

CHAPTER 6

THE SKID FILE

File Description

The record format of the skid file is:

Columns	Length	Format	Item	Data Element
1	1	char(1)		Delete byte
2-29	28	char(28)	1	Key
30	1	char(1)	4	Surface type
31	1	char(1)	6	Curvature
32	1	char(1)	7	Grade
33	1	char(1)	10	Location
34	1	char(1)	11	Surface repair
35-72	38	char(38)	12	Comments
73	1	fixed(1)	5	Surface texture
74	1	fixed(1)	9	Foreign matter
75-76	2	fixed(3)	8	Pavement temperature
77-78	2	fixed(2)	2	Speed
79-80	2	fixed(2)	2	Skid number
81-82	2	fixed(2)		Date of update - month
83-84	2	fixed(2)		Date of update - day
85-86	2	fixed(2)		Date of update - year

The contents of these data elements are described in the publication Highway Information System Release 4.0 - Data Coding Manual. The item numbers listed under the column "Item" are the item numbers used in that manual.

The delete byte in column 1 is used by the system to indicate deleted records. The date of update is added to the record by the update program.

The SKDRDQ Subroutine

SKDRDQ is a "low-level" subroutine for reading the skid file. It provides the basic access functions but does not incorporate the higher-level functions of SKDRD. Its entry points are:

SKDRDQF	Initialization - open the file
SKDRDQR	Set startkey-endkey range
SKDRDQ	Sequential read
SKDRDQX	Direct read - key-high
SKDRDQC	Close the file

SKDRDQF Entry Point - Call SKDRDQF to open the file. Pass a binary fixed(15) value of 2.

SKDRDQR Entry Point - SKDRDQR can be called to limit access to a portion of the file. Pass a char(28) starting key and a char(28) ending key. After the call, the program will have access to only those records having a key equal to or greater than the starting key and equal to or less than the ending key.

SKDRDQ Entry Point - Call SKDRDQ to read a record. Pass a pointer variable. A record is read and its address is returned in the pointer. A null pointer is returned at end of file.

SKDRDQX Entry Point - SKDRDQX provides a key-high direct read capability. Pass a char(28) key and a pointer. The first record in the file that has a key equal to or larger than the passed key is read and its address is returned in the pointer. A null pointer is returned if the key is larger than that of the last record in the file.

SKDRDQC Entry Point - Call SKDRDQC to close the file. No arguments are passed.

Error Messages - If an error is detected, the subroutine prints an error message and issues a user-1ØØ abend.

***** SKDRDQ-Ø1 - SKDRDQ NOT INITIALIZED

The calling program failed to open the file or passed a value other than 2 when opening the file.

***** SKDRDQ-Ø2 - INVALID SKDRDQR CALL - KEYS ARE startkey AND endkey

One of the keys is in an unacceptable format or the startkey is larger than the endkey.

***** SKDRDQ-Ø3 - INVALID SKDRDQX CALL - KEY IS key

The passed key is in unacceptable format.

The SKDRD Subroutine

SKDRD is a "high-level" subroutine for reading the skid file. It automatically handles these parameters from HIS commands:

DATA, START-MILEPOINT, and END-MILEPOINT
CITY
START-DATE and END-DATE
MAINT-DIV
DIRECTION, LANE, and WHEEL
EFF-DATE
X-COORDINATE, Y-COORDINATE, and SQUARE-SIZE
SELECT-DD and SELECT-SIZE
MAX-#-ENTRIES

Either a DATA, a CITY, or a MAINT-DIV parameter must be coded on the command (more than one can be coded). The remaining parameters are optional.

SKDRD has these entry points:

SKDRDF	Initialization - open the file
SKDRD	Read a record
SKDRDC	Close the file

SKDRDF Entry Point - Call SKDRDF to open the file. Pass a binary fixed (15) value of 2 and a binary fixed(15) option code:

Ø	Roadlog records not needed
1	Roadlog records needed

If a value of 1 is passed in the option code, each skid record returned from a SKDRD call will also have the corresponding roadlog record.

SKDRD Entry Point - Call SKDRD to read a record. Pass a skid pointer variable, a roadlog pointer variable, and a fixed(2) variable. A skid record is read and its address is returned in the skid pointer (a null pointer is returned at end of file). If the SKDRDF call requested roadlog records, the corresponding roadlog record is read and its address is returned in the roadlog pointer (a null pointer is returned if roadlog records were not requested). If a MAINT-DIV parameter was specified on the command, the maintenance division of the skid record is returned in the fixed(2) variable (a value of zero is returned otherwise).

SKDRDC Entry Point - Call SKDRDC to close the file. No arguments are passed.

Error Messages - If an error is detected, the subroutine prints an error message. Most of these messages are accompanied by a user-1ØØ abend.

***** SKDRD-Ø1 - X-COORDINATE, Y-COORDINATE, AND SQUARE-SIZE CODED WITHOUT CITY

If these three parameters are used, a CITY parameter must be coded on the command.

***** SKDRD-Ø2 - ERROR IN READING ROADLOG FILE

No roadlog record exists for a Federal Aid skid record.

***** SKDRD-Ø3 - INVALID COMBINATION OF X-COORDINATE, Y-COORDINATE, AND SQUARE-SIZE

If any of these parameters are used, all three must be coded.

***** SKDRD-Ø4 - UNKNOWN CITY NAME

The name coded in a CITY parameter is not a valid city name.

***** SKDRD-Ø5 - NO DATA, MAINT-DIV, OR CITY

One of these parameters must be coded on the command.

***** SKDRD-Ø6 - UNKNOWN MAINT-DIV

The number coded in the MAINT-DIV parameter is not a valid maintenance division number.

***** SKDRD-1ØØ - MAX-#-ENTRIES COMPLETED

This message is informatory only, and indicates that processing was terminated because of a MAX-#-ENTRIES parameter.

The SKDWRQ Subroutine

SKDWRQ is a subroutine for loading the skid file. It has these entry points:

SKDWRQ	Write a record
SKDWRQC	Close the file

SKDWRQ Entry Point - Call SKDWRQ to write a record. Pass a pointer that contains the record's address. Records must be written in order by key.

SKDWRQC Entry Point - Call SKDWRQC to close the file. No arguments are passed.

The SKDUPD Subroutine

SKDUPD is a subroutine for updating the skid file. It has these entry points:

SKDUPDI	Add a record to the file
SKDUPDG	Read a record for subsequent rewrite
SKDUPDR	Rewrite a record
SKDUPDD	Delete a record
SKDUPDC	Close the file

SKDUPDI Entry Point - Call SKDUPDI to insert a record. Pass a pointer that contains the record's address and a binary fixed(15) return code variable. Return codes are:

Ø Successful
4 Unsuccessful - a record already exists with the new record's key

SKDUPDG and SKDUPDR Entry Points - These entry points are used to rewrite a record. Call SKDUPDG to read the record. Pass a pointer variable and a char(28) key. The record is read and its address returned in the pointer. A null pointer is returned if the record does not exist. After a successful read, make any necessary changes to the record and call SKDUPDR. No arguments are passed.

SKDUPDD Entry Point - Call SKDUPDD to delete a record. Pass a char(28) key and a binary fixed(15) return code variable. Return codes are:

Ø Successful
4 Unsuccessful - record does not exist

SKDUPDC Entry Point - Call SKDUPDC to close the file. No arguments are passed.

Error Messages - If an error is detected, the subroutine prints an error message and issues a user-1ØØ abend.

***** I/O ERROR IN SKDUPD DURING function

An I/O error was detected. If function is INSERT, the overflow areas may be full.

***** SKDUPDR CALLED WITHOUT PREVIOUS READ

SKDUPDG must be called before calling SKDUPDR.

***** SKDUPDR - RECORD KEY CHANGED BEFORE REWRITE

The calling program must not alter the key field of a record before rewriting.

***** SKDUPD CALLED BEFORE REWRITING RECORD - ENTRY POINT name

SKDUPDG was called to read a record but SKDUPDR was not called.

The SKDCMPR Subroutine

SKDCMPR is used by SKDRDQ, SKDWRQ, and SKDUPD for compressing and expanding records when the file format is compressed. The only entry point is SKDCMPR. Pass these arguments:

1. Option code - binary fixed(15)
2. Address of uncompressed record/key - pointer
3. Address of compressed record/key - pointer

Pass one of these values in the option code:

Ø Compress a key
1 Expand a key
2 Compress a record
3 Expand a record

Field lengths are:

Compressed key	char(17)
Expanded key	char(28)
Compressed record	char(7Ø)
Expanded record	char(86)

The SKDDCB Subroutine

SKDDCB is used by SKDRDQ, SKDWRQ, and SKDUPD for opening and closing the skid file. The subroutine checks whether the file is in compressed or uncompressed format.

When calling SKDDCB, pass a binary fixed(15) option code, a binary fixed(15) return code, and the skid DCB. The option code designates the function to be performed:

Ø Close the DCB
1 Open the DCB for QISAM input
2 Open the DCB for QISAM output
3 Open the DCB for BISAM update

Return codes are:

Ø Compressed format
1 Uncompressed format

The return code is not set when closing a DCB.

When opening a DCB for output, the following DCB fields are filled in by SKDDCB:

Field	Compressed	Uncompressed
LRECL	7Ø	86
BLKSIZE	721Ø	7138
RECFM	FB	FB
KEYLEN	17	28

The SKDOPT Subroutine

SKDOPT prints a summary of the parameters coded on a skid command. No arguments are passed.

The SKDCVT Subroutine

SKDCVT can convert a skid data card into record format and vice versa. Pass a pointer that contains the record address, a pointer that contains the data card address, and a binary fixed(15) option code:

Ø Convert record to data card
1 Convert data card to record

Allow 81 characters for a data card and 86 characters for a record.

CHAPTER 7

THE SUFFICIENCY FILES

File Description - Sufficiency File

The record format of the sufficiency file is:

Columns	Length	Format	Item	Data Element
1	1	char(1)		Delete byte
2-16	15	char(15)	1	Key
17-34	18	char(18)	2	Description
35-36	2	fixed(2)	4	Design speed
37	1	fixed(1)	5	Terrain
38-39	2	fixed(2)	6	Average speed
4Ø-41	2	fixed(2)	7	Percent of sight distance less than design
42-43	2	fixed(2)	8	Number of stopping distances less than design
44-45	2	fixed(2)	9	Number of curves sharper than design degree of curvature
46	1	fixed(1)	1Ø	Number of narrow bridges
47-48	2	fixed(2)	11	Foundation rating
49-5Ø	2	fixed(2)	12	Surface rating
51-52	2	fixed(2)	13	Drainage rating
53-56	4	fixed(7,3)	14	Section length
57-58	2	fixed(2)	15	Effective date - month
59-6Ø	2	fixed(2)	15	Effective date - day
61-62	2	fixed(2)	15	Effective date - year
63-64	2	fixed(2)		Date of update - month
65-66	2	fixed(2)		Date of update - day
67-68	2	fixed(2)		Date of update - year

The contents of these data elements are described in the publication Highway Information System Release 4.Ø - Data Coding Manual. The item numbers listed under the column "Item" are the item numbers used in that manual.

The delete byte in column one is used by the system to indicate deleted records. The date of update field is added to the record by the update program.

The rural/urban code of the data card (item 3) is not stored in the record. This information is obtained from the roadlog file when needed.

File Description - Sufficiency Report File

The record format of the sufficiency report file is:

Columns	Length	Format	Contents
1	1	char(1)	Delete byte
2-16	15	char(15)	Key
17	1	char(1)	Remark
18-35	35	char(35)	Description
36-37	2	fixed(2)	County number
38-39	2	fixed(2)	Financial district
40-41	2	fixed(2)	Year built
42-43	2	fixed(2)	Year improved
44-45	2	fixed(2)	Surface width
46-47	2	fixed(2)	Roadway width
48-50	3	char(3)	Surface type
51-54	4	fixed(7,3)	Section length
55-57	3	fixed(5)	ADT
58-60	3	fixed(5)	Design hour volume
61-62	2	fixed(2)	Percent commercial
63-65	3	fixed(4)	Service volume
66-67	2	fixed(2)	Number of accidents
68-69	2	fixed(2)	Foundation rating
70-71	2	fixed(2)	Surface rating
72-73	2	fixed(2)	Drainage rating
74-75	2	fixed(2)	Safety rating
76-77	2	fixed(2)	Capacity rating
78-79	2	fixed(3)	Total rating
80-81	2	fixed(3)	Adjusted rating
82-84	3	fixed(5,3)	Deficient mileage
85-86	2	fixed(2)	Design speed
87	1	fixed(1)	Terrain
88-89	2	fixed(2)	Average speed
90-91	2	fixed(2)	Sight distance
92-93	2	fixed(2)	Stopping distance
94-95	2	fixed(2)	Number of curves
96	1	fixed(1)	Number of narrow bridges
97	1	fixed(1)	Number of lanes
98	1	fixed(1)	Divided/undivided code
99-100	2	fixed(3)	City number
101-103	3	fixed(5)	Current year ADT
104-105	2	fixed(2)	Effective date - month
106-107	2	fixed(2)	Effective date - day
108-109	2	fixed(2)	Effective date - year

The SUFRDQ Subroutine

SUFRDQ is a "low-level" subroutine for reading the sufficiency file. It provides the basic access functions but not the higher-level functions such as the select capability implemented in the SUFRD subroutine.

SUFRDQ has the following entry points:

SUFRDQF	Initialization - open the file
SUFRDQR	Set startkey-endkey range
SUFRDQ	Sequential read
SUFRDQX	Direct read - key-high
SUFRDQP	Position - key-high
SUFRDQC	Close the file
SUFRDQI	Combines functions of SUFRDQF and SUFRDQR

SUFRDQF Entry Point - Call SUFRDQF to open the file. Pass a binary fixed(15) value of 2.

SUFRDQR Entry Point - SUFRDQR can be called to limit access to a portion of the file. Pass a char(15) starting key and a char(15) ending key. After the call, access is limited to those records having a key equal to or larger than the starting key and equal to or smaller than the ending key.

SUFRDQ Entry Point - Call SUFRDQ to read a record. Pass a pointer variable. A record is read and its address is returned in the pointer. At end of file, a null value is returned.

SUFRDQX Entry Point - SUFRDQX performs a direct read operation. Pass a char(15) key and a pointer variable. The first record in the file with a key equal to or larger than the passed key is read and its address returned in the pointer. A null pointer is returned if the passed key is larger than that of the last record in the file.

SUFRDQP Entry Point - SUFRDQP performs a key-high search without actually reading a record. Pass a char(15) key. A SUFRDQP call followed by a SUFRDQ call is equivalent to a SUFRDQX call.

SUFRDQC Entry Point - Call SUFRDQC to close the file. No arguments are passed.

SUFRDQI Entry Point - A call to SUFRDQI is equivalent to a call to SUFRDQF followed by a call to SUFRDQR. Pass a binary fixed(15) value of 2, a char(15) starting key, and a char(15) ending key.

Error Messages -

***** SUFRDQ-Ø1 - SUFRDQ NOT PROPERLY INITIALIZED *****

The calling program failed to open the file or passed a value other than 2 when opening the file. This message is accompanied with a user-1ØØ abend.

The SUFRD Subroutine

SUFRD is a "high-level" subroutine for reading the sufficiency file. It should be used by programs that print listings or summaries from the sufficiency file so that the select capabilities will be implemented. Only one high-level subroutine can be called by any one program.

SUFRD automatically handles the following parameters from HIS commands without intervention from the calling program:

DATA, START-MILEPOINT, and END-MILEPOINT
SELECT-DD and SELECT-SIZE
MAX-#-ENTRIES

A DATA parameter must be coded on the command. The remaining parameters are optional.

SUFRD has the following entry points:

SUFRDF	Initialization - open the file
SUFRD	Sequential read
SUFRDC	Close the file

SUFRDF Entry Point - Call SUFRDF to open the file. Pass a binary fixed(15) value of 2.

SUFRD Entry Point - Call SUFRD to read a record. Pass a pointer variable. A record is read and returned in the pointer. At end of file, a null pointer is returned.

SUFRDC Entry Point - Call SUFRDC to close the file. No arguments are passed.

Error Messages -

***** SUFRD-Ø1 - MAX-#-ENTRIES COMPLETED

This message is informatory only, and indicates that processing was terminated because of a MAX-#-ENTRIES parameter.

***** SUFRD-Ø2 - NO DATA PARAMETER *****

The DATA parameter was omitted from the command. A user-1ØØ abend accompanies this message.

The SFRRDQ Subroutine

SFRRDQ is a "low-level" subroutine for reading the sufficiency report file. It has these entry points:

SFRRDQF	Initialization - open the file
SFRRDQR	Set startkey-endkey range
SFRRDQ	Sequential read
SFRRDQX	Direct read - key-high
SFRRDQP	Position - key-high
SFRRDQC	Close the file
SFRRDQI	Combines functions of SFRRDQF and SFRRDQR

Calling sequences are identical as to the corresponding entry points of SUFRDQ, described above.

The SREPSRT Subroutine

SREPSRT reads the sufficiency report file and returns records sequentially in order by either financial district or by rating. Pass the following arguments:

1. Type code - binary fixed (15):
   1 Sort by financial district
   2 Sort by rating
2. Instruction address - pointer (obtained by calling SETINST).
3. Pointer variable.

The record is returned in the pointer. A null pointer is returned at end of file.

The PRNTSFR Subroutine

PRNTSFR converts a sufficiency report record to character format and prints it (one line is printed per call). Pass a pointer that contains the address of the record being printed.

The SFRWRQ Subroutine

SFRWRQ is a subroutine for loading the sufficiency report file. Entry points are:

SFRWRQ	Write a record
SFRWRQC	Close the file

SFRWRQ Entry Point - Call SFRWRQ to write a record. Pass a pointer that contains the record's address. Records must be written in order by key.

SFRWRQC Entry Point - Call SFRWRQC to close the file. No arguments are passed.

The SUFINB Subroutine

SUFINB is a subroutine for updating the sufficiency file. It can be used to add records to the file and to perform new-key functions. Its entry points are:

SUFINB	Add a record to the file
SUFINBN	Change a record's key
SUFINBC	Close the file

Calling sequences are identical to those of RLGINB, described in chapter 5.

The SUFRWB Subroutine

SUFRWB is a subroutine for updating the sufficiency file. It can be used to rewrite and delete records. Its entry points are:

SUFRWB	Read a record for subsequent rewrite or delete
SUFRWBR	Rewrite a record
SUFRWBD	Delete a record
SUFRWBC	Close the file

Calling sequences are identical to those of RLGRWB, described in chapter 5.

The SUFCVT Subroutine

SUFCVT can convert sufficiency data cards into record format and vice versa. Its entry points are:

SUFCVT	Convert a data card to record format
SUFCVTA	Convert a record to data card format

SUFCVT Entry Point - Pass the char(68) record, the char(8Ø) data card, and a binary fixed(15) return code. Return codes are:

Ø	Successful
1	Unsuccessful - non-numeric character in numeric field

SUFCVTA Entry Point - Pass the char(68) record and the char(8Ø) data card.

Error Messages -

***** (E) CONVERSION ERROR - NON-NUMERIC CHARACTER IN NUMERIC FIELD

This message is printed when a return code of 1 is returned.

CHAPTER 8

THE TRAFFIC FILES

File Description - Traffic File

Traffic records are stored in the following format:

Columns	Length	Format	Item	Data Element
1	1	char(1)		Delete byte
2-16	15	char(15)	2	Key
17	1	char(1)	3	Actual/estimated code
18	1	char(1)	7	Remark
19-2Ø	2	fixed(2)	4-a	Year - oldest year
21-23	3	fixed(5)	4-b	AADT - oldest year
24-25	2	fixed(3,3)	4-c	Out of state - oldest year
26-27	2	fixed(3,3)	4-c	Pickups - oldest year
28-29	2	fixed(3,3)	4-c	Commercial - oldest year
3Ø-31	2	fixed(2)	4-a	Year - second year
32-34	3	fixed(5)	4-b	AADT - second year
35-36	2	fixed(3,3)	4-c	Out of state - second year
37-38	2	fixed(3,3)	4-c	Pickups - second year
39-4Ø	2	fixed(3,3)	4-c	Commercial - second year
41-42	2	fixed(2)	4-a	Year - third year
43-45	3	fixed(5)	4-b	AADT - third year
46-47	2	fixed(3,3)	4-c	Out of state - third year
48-49	2	fixed(3,3)	4-c	Pickups - third year
5Ø-51	2	fixed(3,3)	4-c	Commercial - third year
52-53	2	fixed(2)	4-a	Year - fourth year
54-56	3	fixed(5)	4-b	AADT - fourth year
57-58	2	fixed(3,3)	4-c	Out of state - fourth year
59-6Ø	2	fixed(3,3)	4-c	Pickups - fourth year
61-62	2	fixed(3,3)	4-c	Commercial - fourth year
63-64	2	fixed(3,3)	5	Future factor
65-66	2	fixed(3,3)	6	Design hour volume
67-68	2	fixed(2)	8	Effective date - month
69-7Ø	2	fixed(2)	8	Effective date - day
71-72	2	fixed(2)	8	Effective date - year
73-74	2	fixed(2)		Date of update - month
75-76	2	fixed(2)		Date of update - day
77-78	2	fixed(2)		Date of update - year
79-8Ø	2			Unused at this time

The contents of these data elements are described in the publication Highway Information System Release 4.Ø - Data Coding Manual. The item numbers listed under the column "Item" are the item numbers used in that manual.

All data elements are stored in standard IBM character or packed decimal (fixed) formats. For character formats, the length specification given above is the length in characters. For fixed formats, the first or only length specification is the

length in digits. The second specification is the number of those digits that are treated as being to the right of an assumed decimal point.

When a record is deleted from the file, it is not physically removed until the file is reorganized. The delete byte in column one is used by the system to indicate deleted records.

The date of update field is added to the record automatically by the update program.

The first record in the file contains a summary of the years stored in the file. The key of this record is AØØØØØ followed by nine blanks. The record format is:

Columns	Length	Format	Contents
1	1	char(1)	Delete byte
2-16	15	char(15)	Key - AØØØØØ and nine blanks
17-18	2	char(2)	Year - oldest year
19-2Ø	2	char(2)	Year - second year
21-22	2	char(2)	Year - third year
23-24	2	char(2)	Year - fourth year
25-8Ø	56		Unused at this time

File Description - Traffic Report File

Traffic report records are stored in the following format:

Columns	Length	Format	Contents
1	1	char(1)	Delete byte
2-16	15	char(15)	Key
17	1	char(1)	Remark
18-21	4	fixed(7,3)	Section length
22-27	6	fixed(11,3)	Vehicle miles - oldest year
28-33	6	fixed(11,3)	Out of state vm - oldest year
34-39	6	fixed(11,3)	Pickups - oldest year
4Ø-45	6	fixed(11,3)	Commercial vm - oldest year
46-51	6	fixed(11,3)	Vehicle miles - second year
52-57	6	fixed(11,3)	Out of state vm - second year
58-63	6	fixed(11,3)	Pickups vm - second year
64-69	6	fixed(11,3)	Commercial vm - second year
7Ø-75	6	fixed(11,3)	Vehicle miles - third year
76-81	6	fixed(11,3)	Out of state vm - third year
82-87	6	fixed(11,3)	pickups vm - second year
88-93	6	fixed(11,3)	Commercial vm - third year
94-96	3		Unused at this time

This file contains vehicle miles for sections located between two major section breaks of the traffic file. It is software-generated by the program CREATE-TRAFREP.

The following remark codes are used:

W	Rural section
T	Municipal section
O	Out of state section
N	Non-existent section
C	Coincident descriptor
S	Spur descriptor
L	Loop descriptor
D	End-of-section descriptor (route discontinuity)
E	End-of-route and end-of-system descriptors

An E record with rural route totals is stored at the end of each route. Its key is "rrrrrr999RURAL" followed by one blank (rrrrrr is the route system and number). Another E record with rural system totals is stored at the end of each route system. Its key is "s99999RURAL" followed by four blanks, where s is the system code.

The first record in the file indicates the years for which data is currently stored. The key of this record is AØØØØØ followed by nine blanks. The format of this record is:

Columns	Length	Format	Contents
1	1	char(1)	Delete byte
2-16	15	char(15)	Key - AØØØØØ plus nine blanks
17-18	2	char(2)	Year - oldest year
19-2Ø	2	char(2)	Year - second year
21-22	2	char(2)	Year - third year
23-96	74		Unused at this time

The TRFRDQ Subroutine

TRFRDQ is a "low-level" subroutine for reading the traffic file sequentially. It provides the basic access functions, but does not incorporate higher-level functions such as those implemented in TRFRD.

TRFRDQ has these entry points:

TRFRDQF	Initialization - open the file
TRFRDQR	Set startkey-endkey range
TRFRDQ	Sequential read
TRFRDQX	Direct read - key-high
TRFRDQL	Direct read - key-low
TRFRDQP	Position - key-high
TRFRDQT	Set record type
TRFRDQC	Close the file

TRFRDQ1	Save current environment
TRFRDQ2	Restore current environment
TRFRDQI	Combines functions of TRFRDQF and TRFRDQR

<u>TRFRDQF Entry Point</u> - Call TRFRDQF to open the file. Pass a binary fixed(15) value of 2.

<u>TRFRDQR Entry Point</u> - Call TRFRDQR to specify file limits. Pass a char(15) starting key and a char(15) ending key.

<u>TRFRDQ Entry Point</u> - Call TRFRDQ to read a record. Pass a pointer variable. The record address is returned in the pointer. A null value is returned at end of file.

<u>TRFRDQX Entry Point</u> - Call TRFRDQX to read direct using the key-high function. Pass a char(15) key and a pointer. The record address is returned in the pointer. A null value is returned if the key is larger than that of the last record in the file.

<u>TRFRDQL Entry Point</u> - Call TRFRDQL to read direct using the key-low function. Pass a char(15) key and a pointer. The record address is returned in the pointer. A null value is returned if (1) the route does not exist, (2) the key is smaller than that of the first record stored for the route, or (3) the key is larger than that of the last record stored for the route. Use of this entry point may substantially increase execution time.

<u>TRFRDQP Entry Point</u> - Call TRFRDQP to position key-high. Pass a char(15) key. A call to TRFRDQP followed by a call to TRFRDQ is equivalent to a call to TRFRDQX.

<u>TRFRDQT Entry Point</u> - Call TRFRDQT to select records based on type. Pass the sum of the desired values in a binary fixed(15) variable:

1	Major section break records
2	Minor section break records
4	Descriptor records

TRFRDQC Entry Point - Call TRFRDQC to close the file. No arguments are passed.

TRFRDQ1 and TRFRDQ2 Entry Points - These entry points are used by the select routines to enable access to the traffic files without disrupting usage by the mainline program.

TRFRDQI Entry Point - A call to TRFRDQI is equivalent to a call to TRFRDQF followed by a call to TRFRDQR. Pass a binary fixed(15) value of 2, a char(15) starting key, and a char(15) ending key.

Error Messages - If an error is detected, the subroutine prints an appropriate message and issues a user-1ØØ abend.

```
***** TRFRDQ-Ø1 - TRFRDQ NOT PROPERLY INITIALIZED -
      ENTRY POINT entry-point-name

      The calling program either did not open the file or passed a
      value other than 2.

***** TRFRDQ-Ø2 - INVALID TRFRDQ1 CALL

      TRFRDQ1 was called twice without an intervening TRFRDQ2 call.

***** TRFRDQ-Ø3 - INVALID TRFRDQ2 CALL

      TRFRDQ2 was called without an intervening TRFRDQ1 call.
```

The TRFRD Subroutine

TRFRD is a "high-level" subroutine for reading the traffic file. It should be used by programs that print listings or summaries of traffic data so that the select capabilities will be implemented. It should not be used by programs that use the traffic file as a subsidiary file. Only one high-level subroutine can be called by any one program.

TRFRD automatically handles the following parameters from HIS commands without invervention from the calling program:

DATA, START-MILEPOINT, and END-MILEPOINT
TRF-TYPES
SELECT-DD and SELECT-SIZE
MAX-#-ENTRIES

A DATA parameter is required when running a program that uses TRFRD. The other parameters are optional.

TRFRD has these entry points:

TRFRDF	Initialization - open the file
TRFRDT	Set record types
TRFRD	Sequential read
TRFRDX	Direct read - key-high
TRFRDC	Close the file

TRFRDF Entry Point - Call TRFRDF to open the file. Pass a binary fixed(15) value of 2 and a pointer. The pointer must either be null or be set to the address of an 8-character area. If the pointer is not null, the traffic header record (key AØØØØØ) is read and columns 17-24 are placed into the 8-character area provided by the calling program.

TRFRDT Entry Point - Call TRFRDT to select records based on type. Pass the sum of the desired values in a binary fixed(15) variable:

1	Major section break records
2	Minor section break records
4	Descriptor records

If TRFRDT is not called, a default value of 7 is assumed.

TRFRD Entry Point - Call TRFRD to read a record. Pass a pointer variable. The record's address is returned in the pointer. A null value is returned in the pointer at end of file.

TRFRDX Entry Point - Call TRFRDX to read direct (key-high). Pass a char(15) key and a pointer. A null value is returned if the key is higher than any record in the file.

TRFRDC Entry Point - Call TRFRDC to close the file. No arguments are passed.

Error Messages - When an error is detected, the subroutine prints an appropriate message.

***** TRFRD-Ø1 - MAX-#-ENTRIES COMPLETED *****

This message is informatory only, and indicates that an endfile return was performed because the specified number of records had been processed.

***** TRFRD-Ø2 - NO DATA PARAMETER *****

No DATA parameter was coded on the command. A user-1ØØ abend is issued.

The ADT Subroutine

ADT calculates the ADT at a milepoint. If a traffic section break exists at the milepoint, the ADT values are returned directly from the traffic record. Otherwise, an interpolation is performed from the counts preceding and following the milepoint.

ADT entry points include:

ADTE	Set error condition
ADTY	Get years of traffic data
ADT	Calculate ADT
ADTC	Close the traffic and true mileage files

ADTE Entry Point - Standard action in the event of an error is to issue a user-1ØØ abend. To suppress abend action, call ADTE and pass a binary fixed(15) value of 1.

ADTY Entry Point - Call ADTY to determine the years of data being processed. Pass a char(6) variable. Columns 17-22 of the header record (key AØØØØØ) are returned.

ADT Entry Point - Call ADT to retrieve the ADT at a milepoint. Pass the char(15) key, a (4) fixed(5) array, and a binary fixed(15) return code. The ADT for the oldest year is returned in the first array element, the ADT for the second year in the second element, the ADT for the third year in the third element, and the three-year average in the fourth. If data is not available for a year, a zero value is returned. Return codes are:

Ø No data stored
1 Data stored for first year
2 Data stored for second year
3 Data stored for first and second year
4 Data stored for third year
5 Data stored for first and third year
6 Data stored for second and third year
7 Data stored for all three years
8 Error - request could not be completed

ADTC Entry Point - Call ADTC to close the file. No arguments are passed.

The VEHMILE Subroutine

VEHMILE returns the total vehicle miles and section length for a contiguous segment of a route. The calling program can easily compute the section ADT by dividing the vehicle miles by the section length.

VEHMILE has these entry points:

VEHMILE	Compute vehicle miles
VEHMILC	Close the files

VEHMILE Entry Point - To compute vehicle miles, call VEHMILE and pass the following arguments:

1 - char(15) starting key
2 - char(15) ending key
3 - (3) fixed(11,3) array for vehicle miles
4 - (3) char(1) array for error flags
5 - fixed(7,3) variable for section length
6 - pointer for highest DHV record

The two keys must specify the same route and must specify a contiguous section (the section cannot contain any traffic descriptor records). The flag array serves as return codes, and indicates the contents of the corresponding element of the vehicle miles array:

blank	Vehicle miles successfully computed
B	Blank data - stored as zero
X	Error - stored as zero

If the calling program computes a three-year average, the program must check for blank data. The 6th parameter is used if the program wants access to the record having the highest DHV in the section. Pass the address of an 8Ø-byte

area in the pointer. If the highest DHV record is not needed, pass a null pointer.

VEHMILC Entry Point - Call VEHMILC to close the files. No arguments are passed.

Error Messages - If an error is detected, VEHMILE prints an error message and places an X in each element of the flag array.

***** INVALID KEYS PASSED TO VEHMILE SUBROUTINE - startkey AND endkey - KEYS ARE ON DIFFERENT ROUTES

The passed keys were not located on the same route.

***** INVALID KEYS PASSED TO VEHMILE SUBROUTINE - startkey AND endkey - ENDKEY IS BEYOND END OF ROUTE

The endkey is located beyond the end of the route.

***** NULL TRAFFIC POINTER RETURNED TO VEHMILE - KEYS PASSED ARE startkey AND endkey

Data could not be accessed from the traffic file. The keys may specify a route that is not stored in the traffic file.

***** DESCRIPTOR TRAFFIC RECORD RETURNED TO VEHMILE - KEYS PASSED ARE startkey AND endkey - TRAFFIC KEY AND REMARK ARE key remark

A traffic S, L, or C record was located in the section between the startkey and the endkey.

Sample Program - This sample program uses VEHMILE to retrieve vehicle miles and calculates a three-year section ADT from the values returned by VEHMILE.

```
/* SAMPLE PROGRAM - USE OF VEHMILE */

SAMPLE:  PROCEDURE OPTIONS (MAIN);

DECLARE (STARTKEY,ENDKEY) CHAR(15),
        VEHMILES(3) FIXED(11,3),
        FLAG(3)     CHAR(1),
        LENGTH      FIXED(7,3),
        TOTVM       FIXED(13,3),
        AVGVM       FIXED(11,3),
        AVGADT      FIXED(5),
        #YEARS      BIN FIXED,
        PTR_DHV     POINTER,
        VEHMILE     ENTRY,
        VEHMILC     ENTRY;
```

```
          STARTKEY = value;
          ENDKEY   = value;
          PTR_DHV  = NULL();
          CALL VEHMILE (STARTKEY,ENDKEY,VEHMILES,FLAG,LENGTH,PTR_DHV);
          IF FLAG(1)='X' THEN GOTO ERROR;  /* CHECK FOR ERROR */
          TOTVM = VEHMILES(1) + VEHMILES(2) + VEHMILES(3);
          #YEARS = Ø;   /* CHECK FOR BLANK DATA */
          DO I=1 TO 3;
             IF FLAG(I)=' ' THEN #YEARS = #YEARS + 1;
             END;
          IF #YEARS=Ø
             THEN AVGVM = Ø;
             ELSE AVGVM = TOTVM/#YEARS;   /* 3-YEAR AVERAGE VM */
          AVGADT = AVGVM/LENGTH + Ø.5;   /* 3-YEAR AVG ADT - ROUNDED */
                   .
                   .
                   .
      ERROR:
                   .
                   .
                   .
      TERMINATE:
          CALL VEHMILC;   /* CLOSE FILES */
          END SAMPLE;
```

The TRRRDQ Subroutine

TRRRDQ is a subroutine for reading the traffic report file sequentially. It is a low-level read subroutine, similar to TRFRDQ. Select capabilities have not yet been implemented for the traffic report file, so no high-level subroutine exists.

TRRRDQ has these entry points:

TRRRDQF	Initialize - open the file
TRRRDQR	Set startkey-endkey range
TRRRDQ	Sequential read
TRRRDQX	Direct read - key-high
TRRRDQP	Position - key-high
TRRRDQC	Close the file
TRRRDQI	Combines functions of TRRRDQF and TRRRDQR

TRRRDQF Entry Point - Call TRRRDQF to open the file. Pass a binary fixed(15) value of 2.

TRRRDQR Entry Point - Call TRRRDQR to set a startkey-endkey range. Pass a char(15) starting key and a char(15) ending key.

TRRRDQ Entry Point - Call TRRRDQ to read a record. Pass a pointer variable. The record's address is returned in the pointer. A null pointer is returned at end of file.

TRRRDQX Entry Point - Call TRRRDQX to read direct (key-high). Pass a char(15) key and a pointer. A null pointer is returned if the key is larger than that of the largest key in the file.

TRRRDQP Entry Point - Call TRRRDQP to position key-high. Pass a char(15) key. Calling TRRRDQX is equivalent to calling TRRRDQP and then TRRRDQ.

TRRRDQC Entry Point - Call TRRRDQC to close the file. No arguments are passed.

TRRRDQI Entry Point - A call to TRRRDQI is equivalent to a call to TRRRDQF followed by a call to TRRRDQR. Pass a binary fixed(15) value of 2, a char(15) starting key, and a char(15) ending key.

Error Messages - If an error is detected, a message is printed and a user-1ØØ abend is issued.

***** TRRRDQ-Ø1 - TRRRDQ NOT PROPERLY INITIALIZED - ENTRY POINT name
Either TRRRDQF was not called or a value other than 2 was passed.

The TRFINB Subroutine

TRFINB is a subroutine for updating the traffic file. It can add records to the file and perform new-key operations. It has these entry points:

TRFINB	Add a record to the file
TRFINBN	Change a record's key
TRFINBC	Close the file

Calling sequences and results are the same as for RLGINB (see chapter 5).

The TRFRWB Subroutine

TRFRWB is a subroutine for updating the traffic file. It can rewrite and delete records. It has these entry points:

TRFRWB	Read a record for subsequent rewrite or delete
TRFRWBR	Rewrite a record
TRFRWBD	Delete a record
TRFRWBC	Close the file

Calling sequences and results are the same as for RLGRWB (see chapter 5).

The TRRWRQ Subroutine

TRRWRQ is a subroutine for loading the traffic report file. It has these entry points:

TRRWRQ	Write a record
TRRWRQC	Close the file

TRRWRQ Entry Point - Call TRRWRQ to write a record. Pass a pointer that contains the address of the record. Records must be written in order by key.

TRRWRQC Entry Point - Call TRRWRQC to close the file. No arguments are passed.

The TRFCVT Subroutine

TRFCVT can convert traffic "data strings" into traffic record format and vice versa. A data string is a concatenation of traffic data cards used by the update program, and has this format:

Columns	Length	Format	Contents
1	1	char(1)	Unused
2-16	15	char(15)	Key
17	1	char(1)	Actual/estimated code
18-65	48	char(48)	Columns 18-65 of A card
66-81	16	char(16)	Columns 18-33 of B card
82-84	3	char(3)	Future factor
85-87	3	char(3)	Design hour volume
88	1	char(1)	Remark
89-94	6	char(6)	Effective date

TRFCVT has these entry points:

TRFCVT	Convert a data string to record format
TRFCVTA	Convert a record to data string format

<u>TRFCVT Entry Point</u> - Pass a pointer that has the address of an 8∅-character area for storing the record, a pointer that contains the data string address, and a binary fixed(15) return code variable. Return codes are:

∅	Successful
1	Unsuccessful - non-numerics in numeric field

A return code of 1 is accompanied with an error message.

<u>TRFCVTA Entry Point</u> - Pass a pointer that contains the record address, and a pointer that contains the address of a 94-character area for storing the data string.

<u>Error Messages</u> - If a return code of 1 is returned to TRFCVT, this message is printed:

```
***** (E) CONVERSION ERROR - NON-NUMERIC CHARACTER IN NUMERIC FIELD
```

CHAPTER 9

THE TRUE MILEAGE FILE

File Description

The format of true mileage records is:

Columns	Length	Format	Item	Data Element
1	1	char(1)		Delete byte
2-1Ø	9	char(9)	1	Key
11-14	4	fixed(6,3)	2	True mileage
15-16	2	fixed(2)	3	Effective date - month
17-18	2	fixed(2)	3	Effective date - day
19-2Ø	2	fixed(2)	3	Effective date - year
21-22	2	fixed(2)		Date of update - month
23-24	2	fixed(2)		Date of update - day
25-26	2	fixed(2)		Date of update - year

The contents of items 1-3 are described in the publication Highway Information System Release 4.Ø - Data Coding Manual. The data elements are stored in the file in standard IBM character or packed decimal (fixed) formats. For character formats, the length specification shown above (as in char(9)) gives the length in characters. For fixed formats, the first or only length specification gives the length in digits and the second length specification gives the number of these digits that are assumed to lie to the right of an assumed decimal point.

When a record is deleted from the file, it is not physically removed until the file is reorganized. The delete byte in column one is used by the system to indicate deleted records. Any records containing the delete code (X'FF') in this position are ignored by all programs.

The date of update field is added to the record by the update program. This function is automatic and requires no coding.

The TRMRDQ Subroutine

TRMRDQ is a subroutine for reading the true mileage file sequentially. It contains the following entry points:

TRMRDQF	Initialization - open the file
TRMRDQR	Set startkey-endkey range
TRMRDQ	Read one record - sequential access
TRMRDQX	Direct read - key-high

TRMRDQP	Position - key-high
TRMRDQC	Close the file
TRMRDQI	Combines functions of TRMRDQF and TRMRDQR

TRMRDQF Entry Point - Before the calling program can request data from the true mileage file, the subroutine must be initialized. Initialization is accomplished by calling TRMRDQF. Pass a binary fixed (15) value of 2, as in this example:

```
DECLARE TRMRDQF ENTRY (BIN FIXED);
CALL TRMRDQF (2);
```

Initialization can be accomplished by calling TRMRDQI instead of TRMRDQF - see below.

TRMRDQR Entry Point - The calling program can limit access to a portion of the file by calling TRMRDQR. Pass a starting key and an ending key (each key is char(9)). After the call, the program will have access to only those records with a key larger than or equal to the starting key and smaller than or equal to the ending key.

TRMRDQ Entry Point - A call to this entry point returns one record from the file. Records are read sequentially in order by key. Pass a pointer variable. The subroutine will read a record and store its address in the pointer. At end of file, a null value (X'FFØØØØØØ') is stored in the pointer. A call to TRMRDQ is similar to the PL/I statements:

```
ON ENDFILE (TRUMILE) PTR = NULL();
READ FILE (TRUMILE) SET (PTR);
```

TRMRDQX Entry Point - TRMRDQX allows access to the file by key. Pass a char(9) key and a pointer. The subroutine locates the first record in the file that has a key equal to or larger than the passed key and returns its address in the pointer. A null value is returned if the passed key is larger than that of any record in the file.

TRMRDQP Entry Point - This entry point can be used to position key-high to a record in the file. Pass a char(9) key. A TRMRDQP call followed by a TRMRDQ call is equivalent to a TRMRDQX call.

TRMRDQC Entry Point - When the program has finished accessing the true mileage file, it must call TRMRDQC to close the file. No arguments are passed.

TRMRDQI Entry Point - A call to TRMRDQI is equivalent to a call to TRMRDQF followed by a call to TRMRDQR. Pass a binary fixed(15) value of 2, a char(9) starting key, and a char(9) ending key.

Error Messages - If an error is detected, the subroutine prints one of these messages and issues a user-1ØØ abend.

***** TRMRDQ-Ø1 - TRMRDQ NOT PROPERLY INITIALIZED

The calling program either did not initialize the subroutine or passed a value other than 2.

***** TRMRDQ-Ø2 - INVALID TRMRDQR OR TRMRDQI CALL - KEYS ARE startkey AND endkey

The startkey is larger than the endkey.

The TRMRDB Subroutine

TRMRDB is a subroutine for reading the true mileage file directly. It contains the following entry points:

TRMRDBF	Initialization - open the file
TRMRDB	Direct read
TRMRDBC	Close the file

TRMRDBF Entry Point - Call TRMRDBF to open the file. Pass a binary fixed(15) value of 2.

TRMRDB Entry Point - Call TRMRDB to read a record. Pass a pointer and a char(9) key. The pointer is set to the record's address. If the record does not exist, a null value is returned.

TRMRDBC Entry Point - Call TRMRDBC to close the file. No arguments are passed.

Error Messages - If an error is detected, one of these messages is printed and a user-1ØØ abend is issued.

> ***** TRMRDB-Ø1 - TRMRDB NOT PROPERLY INITIALIZED
>
> The calling program either failed to call TRMRDBF or passed a value other than 2.
>
> ***** TRMRDB-Ø2 - I/O ERROR IN TRMRDB
>
> The DECB indicated an I/O error other than record not found.

The TRMRDR Subroutine

TRMRDR is used by POINTQ and DISTQ for reading all of the records for a route. This subroutine is not available for use by other programs.

The true mileage records are read into an array compiled as a separate CSECT named TRMRTE. TRMRTE consists of an 8-character field containing the route number and a (Ø:999) fixed(7,3) array. At any reference post for which no entry is stored in the file, a zero value is stored. If the char(8) field contains blanks, no entries are stored in the array.

To read the true mileage of a route, call TRMRDR passing a char(6) route number. If no records are stored for the route, the message ***** UNKNOWN ROUTE PASSED TO TRMRDR SUBROUTINE - route ***** is printed and the char(8) field of TRMRTE is filled with blanks.

Upon completion, the calling program calls entry point TRMRDRC to close the true mileage file.

The POINTQ and POINTB Subroutines

These subroutines can supply the true mileage at a milepoint, supply the milepoint at a true mileage, and adjust a milepoint to refer to the closest reference post preceding the specified location. Calling sequences to the two subroutines are identical - they differ in the method of accessing the true mileage file. POINTQ has been found to be more efficient for most types of processing. The remainder of this section describes POINTQ but is equally applicable to POINTB.

POINTQ has these entry points:

POINTQO	Initialization
POINTQ	Retrieve true mileage and/or adjust milepoint
POINTQK	Retrieve milepoint
POINTQC	Close the file

POINTQO Entry Point - POINTQO can be called to specify processing options. Pass a binary fixed (15) value ranging from Ø to 7, calculated by adding the appropriate values:

1	Print an error message in the event of an error
2	Issue a user-1ØØ abend in the event of an error
4	Suppress key adjustment during POINTQ calls

If POINTQO is not called, a default value of 7 is assumed.

POINTQ Entry Point - Call POINTQ to retrieve the true mileage at a milepoint. The key is in the format rrrrrrnnn+d.ddd or rrrrrrnnn-d.ddd, where rrrrrr is the route system and number, nnn is the reference post, and ±d.ddd or -d.ddd is the distance from the reference post. The true mileage for the reference post is retrieved and the distance is added. The result is returned. If POINTQO was previously called with a value of 3 or less, the subroutine checks whether the milepoint was already adjusted to the nearest reference posts and performs an adjustment if needed. Parameters passed to POINTQ are a char(15) key and a fixed(6,3) variable for the true mileage. If an error is detected and the POINTQO call did not suppress abends, a value of -1.ØØØ is returned for the true mileage.

POINTQK Entry Point - Call POINTQK if the true mileage is known and the milepoint is desired. Pass a char(15) key and the fixed(6,3) true mileage. The first 6 characters of the key must contain the route system and route number, and the remainder of the key when passed may contain any characters. If an error is detected and abends are suppressed, blanks are returned in the milepoint portion of the key.

POINTQC Entry Point - Call POINTQC to close the file. No arguments are passed.

Error Messages - POINTQ normally prints an error message when an error is detected. The calling program can disable the message function if it prints its own message. The calling program can have the subroutine issue a user-1ØØ abend when an error occurs. Error messages that are currently implemented include:

***** ERROR OCCURRED IN POINTQ ENTRY POINT
***** ERROR OCCURRED IN POINTQK ENTRY POINT

> This message is printed to indicate in which entry point the error was detected. Another message accompanies this message to indicate the nature of the error.

***** INVALID KEY PASSED TO POINTQ SUBROUTINE - FIRST CHARACTER IS BLANK

> The first character of the key is a blank - the key must contain a route system and route number in the first 6 characters.

***** INVALID KEY PASSED TO POINTQ SUBROUTINE - ROUTE DOES NOT EXIST

> No data was stored in the true mileage file for the passed route.

***** INVALID KEY PASSED TO POINTQ SUBROUTINE - REFERENCE POST DOES NOT EXIST

> (POINTQ entry point only) No record was stored in the true mileage file for the passed reference post.

***** INVALID KEY PASSED TO POINTQ SUBROUTINE - TRUE MILEAGE IS NEGATIVE

> The resultant true mileage (as could result from a key such as PØØØØ8 ØØ1-3.5ØØ) is negative.

***** INVALID KEY PASSED TO POINTQ SUBROUTINE - NO TRUE MILEAGE

> The true mileage contains insufficient data for converting a true mileage value to a milepoint or for adjusting a milepoint to the nearest reference post.

The DISTQ and DISTB Subroutines

These subroutines calculate the distance between two milepoints on the same route. The keys passed do not have to be previously adjusted to the nearest reference post and can have positive or negative displacements from the reference post.

Calling sequences and results from DISTQ and DISTB are identical. The two subroutines differ only in the method of accessing the true mileage file. It has been found that DISTQ operates best in almost all cases. The remainder of this section describes DISTQ but is equally applicable to DISTB.

DISTQ entry points include:

DISTQO	Initialization
DISTQ	Calculate distance between two milepoints
DISTQC	Close the file

DISTQO Entry Point - DISTQO can be called to indicate the actions to be performed in the event of an error. Pass a binary fixed(15) value ranging from Ø to 3:

- Ø Return control without printing an error message
- 1 Return control after printing an error message
- 2 Abend without printing an error message
- 3 Abend after printing an error message

If DISTQO is not called, a value of 3 is assumed. All abends are user-1ØØ.

DISTQ Entry Point - Call DISTQ to calculate the distance between two milepoints. Pass two char(15) keys and a fixed(6,3) variable. The two keys must have the same route system and number. The distance is always returned as a positive number regardless of what order the keys are passed. If an error occurs and a value of Ø or 1 was passed to DISTQO, control is returned with a value of -1.ØØØ in the fixed(6,3) variable.

DISTQC Entry Point - Call DISTQC to close the file. No arguments are passed.

Error Messages - If an error occurs, the subroutine can print an error message. The message format is:

```
***** INVALID KEYS PASSED TO DISTQ SUBROUTINE - key,key -
      ERROR CODE IS x *****
```

The error code indicates the nature of the error:

- A Either(1) one of the keys contained a blank as its first character, or (2) the keys specified different routes.
- B The route specified in the keys does not exist in the true mileage file.
- C Either (1) one of the keys specified a non-existent reference post, or (2) the resultant true mileage for one of the keys was negative.

The TRMINB Subroutine

TRMINB is a subroutine for updating the true mileage file. It can be used to add records to the file and to perform new-key functions. It has the following entry points:

TRMINB	Add a record to the file
TRMINBN	Change a record's key
TRMINBC	Close the file

Calling sequences and results are the same as for RLGINB (see chapter 5), except that keys are passed in char(9) format.

The TRMRWB Subroutine

TRMRWB is a subroutine for updating the true mileage file. It can be used to rewrite and delete records in the file. It has the following entry points:

TRMRWB	Read a record for rewrite or delete
TRMRWBR	Rewrite a record
TRMRWBD	Delete a record
TRMRWBC	Close the file

Calling sequences and results are the same as for RLGRWB (see chapter 5), except that keys are passed in char(9) format.

The TRMRWQ Subroutine

TRMRWQ is a subroutine for updating the true mileage file. It can rewrite and delete records. It is designed for sequential processing for the SEQL-REWRITE function of the update program. Entry points are:

TRMRWQ	Set starting point
TRMRWQR	Rewrite
TRMRWQD	Delete
TRMRWQF	Terminate
TRMRWQC	Close the file

TRMRWQ Entry Point - Call TRMRWQ to begin processing a series of records. Pass a char(9) key and a pointer. The specified record is read (key-high access) and its address placed in the pointer. A system Ø31 abend will result if a key higher than the last key in the file is passed.

TRMRWQR Entry Point - Call TRMRWQR to rewrite a record and read the next record. Pass the pointer that was passed to TRMRWQ (its value cannot have been altered). TRMRWQR stores the current date in the date of update and rewrites the record. The calling program must not have altered the key. The next record is read and its address placed into the pointer.

TRMRWQD Entry Point - This entry point is identical to TRMRWQR except that the record is deleted.

TRMRWQF Entry Point - Call TRMRWQF to terminate processing a series of records. No parameters are passed. The last record that was read is not rewritten.

TRMRWQC Entry Point - Call TRMRWQC to close the file. No arguments are passed.

Error Messages - If an error is detected, a message is printed and a user-1ØØ abend is issued. The message is:

***** PROGRAM ERROR - INVALID CALL TO entry-point-name *****

Possible causes are:

1. TRMRWQR or TRMRWQD was called without a preceding TRMRWQ call.

2. TRMRWQR or TRMRWQD was called after a TRMRWQF call without an intervening TRMRWQ call.

3. TRMRWQR or TRMRWQD was called after a null pointer was returned.

4. The calling program altered a record's key or altered a pointer.

The TRMCVT Subroutine

TRMCVT can convert true mileage data cards into record format and vice versa. Entry points are:

TRMCVT	Convert from card format to record format
TRMCVTA	Convert from record format to card format

TRMCVT Entry Point - Pass a pointer that contains the address of a 26-character area into which the record will be placed, a pointer that contains the address of the 8Ø-character card, and a binary fixed(15) return code. The return code is set by TRMCVT to:

Ø Successful
1 Unsuccessful - non-numerics in numeric fields

TRMCVTA Entry Point - Pass a pointer that contains the address of the 26-character record, and a pointer that contains the address of an 8Ø-character area into which the card will be placed.

Error Messages - If a return code of 1 is returned by TRMCVT, the following message is printed:

***** (E) CONVERSION ERROR - NON-NUMERIC CHARACTER IN NUMERIC FIELD

CHAPTER 1Ø

THE URBAN SIGN INVENTORY FILES

File Description - Urban Sign Inventory Files

The urban sign inventory files contain three types of records: assembly records, sign records, and remark records. Assembly records have the following format:

Columns	Length	Format	Item	Data Element
1	1	char(1)		Delete byte
2	1	char(1)	A-2	Assembly number tie-breaker
3-7	5	char(5)	A-3	Assembly number
8	1	char(1)	A-4	Sign number (blank)
9-11	3	char(3)		Sequence number
12-15	4	char(4)	A-1	Update code
16-17	2	fixed(2)	A-5	Month
18-19	2	fixed(2)	A-5	Day
2Ø-21	2	fixed(2)	A-5	Year
22-23	2	fixed(2)	A-5	Hour
24-25	2	fixed(2)		Date of installation - month
26-27	2	fixed(2)		Date of installation - day
28-29	2	fixed(2)		Date of installation - year
3Ø	1	char(1)		Date of installation code
31	1	fixed(1)	A-6	Post condition
32	1	fixed(1)	A-7	Post position
33-46	14	char(14)	A-8	Assembly location
47-49	3	fixed(4)		X-coordinate
5Ø-52	3	fixed(4)		Y-coordinate
53-55	3	char(3)	A-9	Map number
56	1	char(1)	A-1Ø	Assembly type
57-58	2	fixed(2)	A-11	Post type
59	1	fixed(1)	A-12	Number of posts
6Ø	1	fixed(1)	A-13	Visibility
61-62	2	fixed(2)	A-14	Change in mounting height
63-64	2	fixed(2)	A-15	Change in lateral clearance
65-68	4	fixed(6)		X-feet
69-72	4	fixed(6)		Y-feet
73-77	5	char(5)		Unused
78-79	2	fixed(3)	A-16	Cost field 1
8Ø-81	2	fixed(3)	A-16	Cost field 2
82-83	2	fixed(3)	A-16	Cost field 3
84-86	3	fixed(4)	A-16	Cost field 4
87-88	2	fixed(2)		Date of update - month
89-9Ø	2	fixed(2)		Date of update - day
91-92	2	fixed(2)		Date of update - year

Sign records have the following format:

Columns	Length	Format	Item	Data Element
1	1	char(1)		Delete byte
2	1	char(1)	S-2	Assembly number tie-breaker
3-7	5	char(5)	S-3	Assembly number
8	1	char(1)	S-4	Sign number (alphabetic)
9-11	3	char(3)		Sequence number
12-15	4	char(4)	S-1	Update code
16-17	2	fixed(2)	S-5	Month
18-19	2	fixed(2)	S-5	Day
20-21	2	fixed(2)	S-5	Year
22-23	2	fixed(2)	S-5	Hour
24-25	2	fixed(2)		Date of installation - month
26-27	2	fixed(2)		Date of installation - day
28-29	2	fixed(2)		Date of installation - year
30	1	char(1)		Date of installation code
31	1	fixed(1)	S-6	Sign condition
32	1	fixed(1)	S-7	Sign position
33	1	char(1)		Uniform/non-uniform code
34	1	char(1)	S-8	Direction of facing
35-39	5	char(5)	S-9	Visibility on other routes - field 1
40-44	5	char(5)	S-9	Visibility on other routes - field 2
45	1	char(1)	S-10	Side of street
46	1	fixed(1)	S-11	Functional classification
47	1	char(1)	S-12	Federal aid system
48	1	char(1)	S-13	Maintenance responsibility
49-58	10	char(10)	S-14	Sign code number
59-60	2	char(2)	S-15	Supplemental code
61-62	2	fixed(2)	S-16	Sign color
63	1	fixed(1)	S-17	Letter type
64	1	fixed(1)	S-18	Material
65	1	fixed(1)	S-19	Face
66	1	fixed(1)	S-20	Shape
67-68	2	fixed(3)	S-21	Horizontal dimension
69-70	2	fixed(3)	S-21	Vertical dimension
71-72	2	fixed(2)	S-22	Mounting height
73-74	2	fixed(2)	S-23	Lateral clearance
75-77	3	char(3)	S-24	Sign age
78-79	2	fixed(3)	S-25	Cost field 1
80-81	2	fixed(3)	S-25	Cost field 2
82-83	2	fixed(3)	S-25	Cost field 3
84-86	3	fixed(4)	S-25	Cost field 4
87-88	2	fixed(2)		Date of update - month
89-90	2	fixed(2)		Date of update - day
91-92	2	fixed(2)		Date of update - year

Remark records are stored in the following format:

Columns	Length	Format	Item	Data Element
1	1	char(1)		Delete byte
2	1	char(1)	R-2	Assembly number tie-breaker
3-7	5	char(5)	R-3	Assembly number
8	1	char(1)	R-4	Sign number (alphabetic or blank)
9-11	3	char(3)		Sequence number (7ØØ-999)
12-15	4	char(4)	R-1	Update code
16-17	2	fixed(2)	R-5	Month
18-19	2	fixed(2)	R-5	Day
2Ø-21	2	fixed(2)	R-5	Year
22-23	2	fixed(2)	R-5	Hour
24-84	61	char(61)	R-6	Comments
85-86	2	char(2)		Unused
87-88	2	fixed(2)		Date of update - month
89-9Ø	2	fixed(2)		Date of update - day
91-92	2	fixed(2)		Date of update - year

The contents of these data elements are described in the publication Highway Information System Release 4.Ø - Data Coding Manual. The item numbers listed under the column "item" are the item numbers used in that manual.

All data elements are stored in standard IBM character or packed decimal (fixed) formats. The format type and length are shown under the column "format." Character lengths are specified in characters. Fixed lengths are specified in digits.

Notes on Data Elements - Assembly Records

Columns	Item	Notes
1		Delete byte - This byte is used to indicate deleted records.
9-11		Sequence number - Added by update program to ensure unique keys.
22-23	A-5	When hour is specified as 'UN' a value of zero is stored.
24-3Ø		Date of installation - Contains the date from the most recent AINS or AINV record. Date of installation code contains 'S' if the record is AINS or 'V' if the record is AINV.

Columns	Item	Notes
47-52		Coordinates - These are calculated by the update program. They are based on the same system used in the accident subsystem.
65-72		X and Y feet - These are by-products of the x and y coordinates calculations. They are retained because they have more accuracy. They are adjusted to an origin at the lower left corner rather than the origin used for coordinates.
87-92		Date of update - The date is inserted by the update program.

Notes on Data Elements - Sign Records

Columns	Item	Notes
1		Delete byte - This byte is used to indicate deleted records.
9-11		Sequence number - Added by update program to ensure unique keys.
22-23	S-5	Hour - When hour is specified as 'UN' a value of zero is stored.
24-3Ø		Date of installation - Contains the date from the most recent SINS, SINV, or SRPL record. Date of installation code contains 'S' for SINS, 'V' for SINV, or 'R' for SRPL.
33		Uniform/non-uniform code - Contains an '*' if edit checks show that the sign is non-uniform. Contains a blank otherwise.
87-92		Date of update - Added by update program.

Notes on Data Elements - Remark Records

Columns	Item	Notes
1		Delete byte - See sign notes above.
9-11		Sequence number - See sign notes above.
22-23	R-5	Hour - See sign notes above.
87-92		Date of update - See sign notes above.

Data Element Names - Urban Sign Inventory Files

Programs in the Highway Information System that access the Urban sign inventory files use the structures shown in this section. Two modules, HISXUSNRD and HISXSGNRC, are stored in the PANVALET library for use with ++INCLUDE commands. Both modules contain these structure definitions. They are documented later in this chapter.

```
1  ASM,
   2  DLTE_BYTE        CHAR(1),       /* Delete byte */
   2  TIE_BRKR         CHAR(1),       /* Assembly number tie-breaker */
   2  ASMBLY_#         CHAR(5),       /* Assembly number */
   2  SIGN_#           CHAR(1),       /* Sign number */
   2  SEQ_#            CHAR(3),       /* Sequence number */
   2  UPDTE_CD         CHAR(4),       /* Update code */
   2  MONTH            FIXED(2),      /* Month */
   2  DAY              FIXED(2),      /* Day */
   2  YEAR             FIXED(2),      /* Year */
   2  HOUR             FIXED(2),      /* Hour */
   2  MONTH_INS        FIXED(2),      /* Date installed - month */
   2  DAY_INS          FIXED(2),      /* Date installed - day */
   2  YEAR_INS         FIXED(2),      /* Date installed - year */
   2  INS_CODE         CHAR(1),       /* Date installed - code */
   2  POST_COND        FIXED(1),      /* Post condition */
   2  POST_POS         FIXED(1),      /* Post position */
   2  ASMBLY_LOC       CHAR(14),      /* Assembly location */
   2  X_COORD          FIXED(4),      /* X-coordinate */
   2  Y_COORD          FIXED(4),      /* Y-coordinate */
   2  MAP_#            CHAR(3),       /* Map number */
   2  ASMBLY_TYPE      CHAR(1),       /* Assembly type */
   2  POST_TYPE        FIXED(2),      /* Post type */
   2  #_POSTS          FIXED(1),      /* Number of posts */
   2  VIS              FIXED(1),      /* Visibility */
   2  CHNGE_MTNG_HT    FIXED(2),      /* Change in mounting height */
   2  CHNGE_LAT_CLR    FIXED(2),      /* Change in lateral clearance */
   2  X_FEET           FIXED(6),      /* X-feet */
   2  Y_FEET           FIXED(6),      /* Y-feet */
   2  FILLER           CHAR(5),
   2  COST(3)          FIXED(3),      /* Cost fields 1-3 */
   2  COST4            FIXED(4),      /* Cost field 4 */
   2  MONTH_UPDTE      FIXED(2),      /* Date of update - month */
   2  DAY_UPDTE        FIXED(2),      /* Date of update - day */
   2  YEAR_UPDTE       FIXED(2),      /* Date of update - year */
```

```
1  ASMX,
   2  DLTE_BYTE        CHAR(1),      /* Delete byte */
   2  KEY              CHAR(1Ø),     /* Key */
   2  FILLER1          CHAR(21),
   2  RTE_TRAV         CHAR(4),      /* Assembly location - route travelled */
   2  DIR_TRAV         CHAR(1),      /* Assembly location - direction trav */
   2  RTE_INTSECT      CHAR(4),      /* Asm. locn. - route intersected */
   2  DISTANCE         PIC'S9999',   /* Asmbly locn - distance */

1  ASMY,
   2  DLTE_BYTE        CHAR(1),      /* Delete byte */
   2  TIE_BRKR         CHAR(1),      /* Assembly number tie-breaker */
   2  ASMBLY_#         PIC'99999',   /* Assembly number */
   2  SIGN_#           CHAR(1),      /* Sign number */
   2  SEQ_#            PIC'999',     /* Sequence number */

1  SGN,
   2  DLTE_BYTE        CHAR(1),      /* Delete byte */
   2  TIE_BRKR         CHAR(1),      /* Assembly number tie-breaker */
   2  ASMBLY_#         CHAR(5),      /* Assembly number */
   2  SIGN_#           CHAR(1),      /* Sign number */
   2  SEQ_#            CHAR(3),      /* Sequence number */
   2  UPDTE_CD         CHAR(4),      /* Update code */
   2  MONTH            FIXED(2),     /* Month */
   2  DAY              FIXED(2),     /* Day */
   2  YEAR             FIXED(2),     /* Year */
   2  HOUR             FIXED(2),     /* Hour */
   2  MONTH_INS        FIXED(2),     /* Date of installation - month */
   2  DAY_INS          FIXED(2),     /* Date of installation - day */
   2  YEAR_INS         FIXED(2),     /* Date of installation - year */
   2  INS_CODE         FIXED(2),     /* Date of installation code */
   2  SIGN_COND        FIXED(1),     /* Sign condition */
   2  SIGN_POS         FIXED(1),     /* Sign position */
   2  UNIFORM          CHAR(1),      /* Uniform/non-uniform code */
   2  FACE_DIR         CHAR(1),      /* Direction of facing */
   2  VIS_OTHER(2)     CHAR(5),      /* Visibility on other routes */
   2  SIDE             CHAR(1),      /* Side of street */
   2  FUNCT_CLASS      FIXED(1),     /* Functional classification */
   2  FED_AID_SYS      CHAR(1),      /* Federal aid system */
   2  MAINT_RESPN      CHAR(1),      /* Maintenance responsibility */
   2  CODE_#           CHAR(1Ø),     /* Sign code number */
   2  SUPP_CODE        CHAR(2),      /* Supplemental code */
   2  COLOR            FIXED(2),     /* Sign color */
   2  LETTER           FIXED(1),     /* Letter type */
   2  MATL             FIXED(1),     /* Material */
   2  FACE             FIXED(1),     /* Face */
   2  SHAPE            FIXED(1),     /* Shape */
   2  HORIZ_DIM        FIXED(3),     /* Horizontal dimension */
   2  VERT_DIM         FIXED(3),     /* Vertical dimension */
```

```
    2  MTNG_HT        FIXED(2),     /* Mounting height */
    2  LAT_CLR        FIXED(2),     /* Lateral clearance */
    2  AGE            CHAR(3),      /* Sign age */
    2  COST(3)        FIXED(3),     /* Cost fields 1-3 */
    2  COST4          FIXED(3),     /* Cost field 4 */
    2  MONTH_UPDTE    FIXED(2),     /* Date of update - month */
    2  DAY_UPDTE      FIXED(2),     /* Date of update - day */
    2  YEAR_UPDTE     FIXED(2),     /* Date of update - year */

 1  SGNX,
    2  DLTE_BYTE      CHAR(1),      /* Delete byte */
    2  KEY            CHAR(1Ø),     /* Key */
    2  FILLER1        CHAR(63),
    2  AGE_YEARS      PIC'9',       /* Sign age - years */
    2  AGE_MONTHS     PIC'99',      /* Sign age - months */

 1  SGNY,
    2  DLTE_BYTE      CHAR(1),      /* Delete byte */
    2  TIE_BRKR       CHAR(1),      /* Assembly number tie-breaker */
    2  ASMBLY_#       PIC'99999',   /* Assembly number */
    2  SIGN_#         CHAR(1),      /* Sign number */
    2  SEQ_#          PIC'999',     /* Sequence number */
    2  FILLER1        CHAR(63),
    2  AGE            PIC'999',     /* Sign age */

 1  RMK,
    2  DLTE_BYTE      CHAR(1),      /* Delete byte */
    2  TIE_BRKR       CHAR(1),      /* Assembly number tie-breaker */
    2  ASMBLY_#       CHAR(5),      /* Assembly number */
    2  SIGN_#         CHAR(1),      /* Sign number */
    2  SEQ_#          CHAR(3),      /* Sequence number */
    2  UPDTE_CD       CHAR(4),      /* Update code */
    2  MONTH          FIXED(2),     /* Month */
    2  DAY            FIXED(2),     /* Day */
    2  YEAR           FIXED(2),     /* Year */
    2  HOUR           FIXED(2),     /* Hour */
    2  REMARKS        CHAR(61),     /* Remarks */
    2  FILLER         CHAR(2),
    2  MONTH_UPDTE    FIXED(2),     /* Date of update - month */
    2  DAY_UPDTE      FIXED(2),     /* Date of update - day */
    2  YEAR_UPDTE     FIXED(2),     /* Date of update - year */

 1  RMKX,
    2  DLTE_BYTE      CHAR(1),      /* Delete byte */
    2  KEY            CHAR(1Ø),     /* Key */

 1  RMKY,
    2  DLTE_BYTE      CHAR(1),      /* Delete byte */
    2  TIE_BRKR       CHAR(1),      /* Assembly number tie-breaker */
    2  ASMBLY_#       PIC'99999',   /* Assembly number */
    2  SIGN_#         CHAR(1),      /* Sign number */
    2  SEQ_#          PIC'999',     /* Sequence number */
```

The USNRDQ Subroutine

USNRDQ is a "low-level" subroutine for reading the urban sign inventory files. To implement higher-level functions, use the USNRD subroutine.

USNRDQ has the following entry points:

USNRDQF	Initialization - open the file
USNRDQ	Sequential read
USNRDQX	Direct read - key-high
USNRDQC	Termination - close the file

USNRDQF Entry Point - Call USNRDQF to open the file. Pass a fixed(3) city number and a binary fixed(15) return code. Return codes are:

Ø	Successful
1	City number in error
2	DD statement missing
3	Open failed
4	Already open for different city

USNRDQ Entry Point - Call USNRDQ to read a record. Pass a pointer and a binary fixed(15) return code. Return codes are:

Ø	Successful
1	End-of-file
2	File not open
3	Called after end-of-file
4	I/O error

If the return code is zero, the pointer contains the address of the record that has been read.

USNRDQX Entry Point - Call USNRDQX to read direct. Pass a char(1Ø) key, a pointer, and a binary fixed(15) return code. Return codes are:

Ø	Successful
1	Key higher than highest in file
2	File not open
3	
4	I/O error

If the return code is zero, the pointer contains the address of the record that has been read.

USNRDQC Entry Point - Call USNRDQC to close the file. Pass a binary fixed(15) return code. Return codes are:

Ø Successful
1 File not open
2 Close failed

The HISXSGNRC Source Module - This module can be included in a source program by including a ++INCLUDE HISXSGNRC panvalet command in the program. The module contains the following declarations:

```
DECLARE
   PTR_USN POINTER,
   1  ASM BASED(PTR_USN),
        .
        :   (see preceding section for data element names)
   1  SGN BASED(PTR_USN),
        .
        :
   1  RMK BASED(PTR_USN),
        .
        :
   1  ASMX BASED(PTR_USN),
        .
        :
   1  ASMY BASED(PTR_USN),
        .
        :
   1  SGNX BASED(PTR_USN),
        .
        :
   1  SGNY BASED(PTR_USN),
        .
        :
   1  RMKX BASED(PTR_USN),
        .
        :
   1  RMKY BASED(PTR_USN),
        .
        :
      2  SEQ_#      PIC'999';
```

The USNRD Subroutine

USNRD is a "high-level" subroutine for reading the urban sign inventory files. It automatically processes the following command parameters:

START-ASSEMBLY and END-ASSEMBLY
DATE
ACCESS
CITY
MAX-#-ENTRIES
SELECT-DD and SELECT-SIZE

Each call to USNRD returns all of the records that pertain to one sign. The records are returned in a set of four arrays: one each for assembly records, assembly remark records, sign records, and sign remark records.

USNRD contains the following entry points:

USNRDF	Initialization - open the file
USNRD	Sequential read
USNRDC	Termination - close the file

USNRDF Entry Point - Call USNRDF to open the file. Pass a binary fixed(15) return code. Return codes are:

Ø Successful
1 Unsuccessful (accompanied by error message)

USNRD Entry Point - Call USNRD to read a set of records. Pass two pointers and a binary fixed(15) return code. Return codes are:

Ø Successful
1 End-of-file
2 Unsuccessful (accompanied by error message)

If the return code is zero, the two pointers contain addresses of structures. The first pointer points to a structure that indicates how many records are available:

```
1  #REC,
   2  #ASM    BIN FIXED,   /* Number of assembly records */
   2  #SGN    BIN FIXED,   /* Number of sign records */
   2  #ASMRMK BIN FIXED,   /* Number of assembly remark records */
   2  #SGNRMK BIN FIXED,   /* Number of sign remark records */
```

The second pointer points to a structure of records:

```
1  USN_PTRS,
   2  PTR_ASMS     POINTER,  /* Address of assembly array */
   2  PTR_SGNS     POINTER,  /* Address of sign array */
   2  PTR_ASMRMKS  POINTER,  /* Address of assembly remark array */
   2  PTR_SGNRMKS  POINTER,  /* Address of sign remark array */
```

<u>USNRDC Entry Point</u> - Call USNRDC to close the file. Pass a binary fixed(15) return code:

```
Ø  Successful
1  Unsuccessful (accompanied by error message)
```

<u>The HISXUSNRD Source Module</u> - This module is available for use with the panvalet ++INCLUDE command. It contains:

1. Urban sign structure declarations.
2. USNRD entry point declarations.
3. Variable declarations for variables needed in conjunction with USNRD.
4. Procedures that perform USNRD linkages.

To use HISXUSNRD, include a ++INCLUDE HISXUSNRD panvalet command in the source program. To call the USNRD entry points, simply use the following calls:

```
CALL CALL_USNRDF;

CALL CALL_USNRD;

CALL CALL_USNRDC;
```

After any of these calls, interrogate the return code variable USNRD_RC. After a successful call to USNRD, the following variables can be referred to:

```
#REC.#ASM         /* Bin fixed - Number of assembly records */
#REC.#SGN         /* Bin fixed - Number of sign records */
#REC.#ASMRMK      /* Bin fixed - Number of assembly remark records */
#REC.#SGNRMK      /* Bin fixed - Number of sign remark records */

ASMS(1ØØØ)        /* Char(92) - Assembly records */
SGNS(1ØØØ)        /* Char(92) - Sign records */
ASMRMKS(1ØØØ)     /* Char(92) - Assembly remark records */
SGNRMKS(1ØØØ)     /* Char(92) - Sign remark records */

PTR_ASM           /* Pointer - Use to access assembly records */
PTR_SGN           /* Pointer - Use to access sign records */
PTR_RMK           /* Pointer - Use to access remark records */
```

HISXUSNRD has the following structure:

```
DECLARE
   UPDTE_CD(16) CHAR(4) STATIC INITIAL
     ('AREM','SREM','AINV','AINS','SINV','SINS','AREL','ARPL',
      'ARPR','AVIS','ACND','SRPL','SRPR','SRPS','SCND','RMRK'),
   PTR_ASM POINTER,
   1  ASM BASED(PTR_ASM),
         :
         :     (see standard names above)
   1  ASMX BASED(PTR_ASM),
         :
         :
   1  ASMY BASED(PTR_ASM),
         :
         :
   PTR_SGN POINTER,
   1  SGN BASED(PTR_SGN),
         :
         :
   1  SGNX BASED(PTR_SGN),
         :
         :
   1  SGNY BASED(PTR_SGN),
         :
         :
   PTR_RMK POINTER,
   1  RMK BASED(PTR_RMK),
         :
         :
   1  RMKX BASED(PTR_RMK),
         :
         :
   1  RMKY BASED(PTR_RMK),
         :
         :
   PTR_#REC POINTER,
   1  #REC BASED(PTR_#REC),
      2  #ASM    BIN FIXED,
      2  #SGN    BIN FIXED,
      2  #ASMRMK BIN FIXED,
      2  #SGNRMK BIN FIXED,
   PTR_USNPTRS POINTER,
   1  USNPTRS BASED(PTR_USNPTRS),
      2  PTR_ASMS    POINTER,
      2  PTR_SGNS    POINTER,
      2  PTR_ASMRMKS POINTER,
      2  PTR_SGNRMKS POINTER,
```

```
      PTR_ASMS    POINTER,
      PTR_SGNS    POINTER,
      PTR_ASMRMKS POINTER,
      PTR_SGNRMKS POINTER,
      ASMS(1ØØØ)     CHAR(92) BASED(PTR_ASMS),
      SGNS(1ØØØ)     CHAR(92) BASED(PTR_SGNS),
      ASMRMKS(1ØØØ) CHAR(92) BASED(PTR_ASMRMKS),
      SGNRMKS(1ØØØ) CHAR(92) BASED(PTR_SGNRMKS),
      USNRD_RC BINARY FIXED,
      USNRDF   ENTRY (BINARY FIXED),
      USNRD    ENTRY (POINTER,POINTER,BINARY FIXED),
      USNRDC   ENTRY (BINARY FIXED);

   CALL_USNRDF:  PROCEDURE;
      CALL USNRDF (USNRD_RC);
      END CALL_USNRDF;

   CALL_USNRDC:  PROCEDURE;
      CALL USNRDC (USNRD_RC);
      END CALL_USNRDC;

   CALL_USNRD:  PROCEDURE;
      CALL USNRD (PTR_#REC,PTR_USNPTRS,USNRD_RC);
      IF USNRD_RC=Ø THEN DO;
         PTR_ASMS    = USNPTRS.PTR_ASMS;
         PTR_SGNS    = USNPTRS.PTR_SGNS;
         PTR_ASMRMKS = USNPTRS.PTR_ASMRMKS;
         PTR_SGNRMKS = USNPTRS.PTR_SGNRMKS;
         END;
      END CALL_USNRD;
```

The USNXY Subroutine

USNXY calculates x- and y-coordinates and x- and y-feet from an assembly location. Pass the following arguments:

1. Assembly location - char(14).
2. X-coordinate - fixed(4).
3. Y-coordinate - fixed(4).
4. X-feet - fixed(6).
5. Y-feet - fixed(6).

If an error occurs, an error message is printed and zeroes are returned in all of the fixed arguments.

The CALCAGE Subroutine

CALCAGE calculates the age of a sign from the date of installation and the age at time of installation fields. Pass the following arguments:

1. Age from sign record - char(3).
2. Month of installation - fixed(2).
3. Day of installation - fixed(2).
4. Year of installation - fixed(2).
5. Calculated age - char(4).

The calculated age is returned in the format yymm, where yy is the number of years and mm is the number of months. A value of 'UNK ' is returned for any of the following conditions:

1. Age from sign record contains 'UNK'.
2. Age from sign record contains any non-numerics.
3. Month, day, or year of installation is not a valid fixed(2) value.
4. Date of installation is more recent than the computer's current date.

A value of '9999' is returned for either of the following conditions:

1. Age from sign record contains '999'.
2. Calculated age is more than 99 years 11 months.

The age calculations performed are:

1. If current year is the same as the year of installation:

   #-months = age-months + (current-month - month-ins)

   #-years = age-years

2. If current year differs from year of installation:

   #-months = age-months + current-month + (12 - month-ins)

   #-years = age-years + (current-year - year-ins - 1)

The resultant value is normalized so that #-months ranges from Ø to 11. The value shown in these equations are:

#-months - Number of months returned in calculated age.
#-years - Number of years returned in calculated age.

age-months - Number of months passed in stored age.
age-years - Number of years passed in stored age.

month-ins - Month of installation.
year-ins - Year of installation.

current-month - Month portion of computer's current date.
current-year - Year portion of computer's current date.

The USNCVT Subroutine

USNCVT converts urban sign inventory data cards into file records and vice versa. Pass the following arguments:

1. Option code - binary fixed(15).
2. Address of data card - pointer.
3. Address of record - pointer.
4. Return code - binary fixed(15).

Option codes are:

Ø Convert assembly data card to record
1 Convert assembly record to data card
2 Convert sign data card to record
3 Convert sign record to data card
4 Convert remark data card to record
5 Convert remark record to data card

Return codes are:

Ø Successful
1 Unsuccessful - Bad option code
2 Unsuccessful - PL/I error condition raised (probably caused by a non-numeric character in a numeric field).

Assembly Card to Record - When an assembly card is converted to record format, the following steps are taken:

1. All corresponding fields are copied with any necessary data conversions.
2. The sequence number field is set to 'ØØØ'.
3. If the hour is specified as 'UN' a value of zero is stored.
4. If the update code is 'AINS' or 'AINV', the date field is also stored in the date of installation field and an 'S' or 'V' is stored in the date of installation code.
4. If the update code is neither 'AINS' nor 'AINV', the date of installation is set to zeroes and the date of installation code is set to blank.
5. The x-coordinate, y-coordinate, x-feet, and y-feet fields are set to zero.
6. The date of update is set to zeroes.
7. Numeric fields that contain all blanks are copied as zeroes. All other numeric fields must contain leading zeroes.

Assembly Record to Card - When an assembly record is converted to card format, the following steps are taken:

1. All corresponding fields are copied with any necessary data conversions.
2. If the hour is zero, it is copied as 'UN'.
3. All decimal fields that contain zeroes are copied as blanks. All decimal fields that contain non-zeroes are copied with leading zeroes.

Sign Card to Record - When a sign data card is converted to record format, the following steps are taken:

1. All corresponding fields are copied with any necessary data conversions.
2. The sequence number field is set to 'ØØØ'.
3. If the hour is specified as 'UN' a value of zero is stored.
4. If the update code is 'SINS', 'SINV', or 'SRPL', the date field is also stored in the date of installation field and an 'S', 'V', or 'R' is stored in the date of installation code.
5. If the update code is none of the codes in (4), the date of installation is set to zeroes and the date of installation code is set to blank.
6. The uniform/non-uniform code is set to blank.
7. The date of update is set to zeroes.
8. Numeric fields that contain all blanks are copied as zeroes. All other numeric fields must contain leading zeroes.

Sign Record to Card - When a sign record is converted to data card format, the following steps are taken:

1. All corresponding fields are copied with any necessary data conversions.
2. If the hour is zero it is copied as 'UN'.
3. All decimal fields that contain zeroes are copied as blanks. All decimal fields that contain non-zeroes are copied with leading zeroes.

<u>Remark Card to Record</u> - When a remark data card is converted to record format, the following steps are taken:

1. All corresponding fields are copied with any necessary data conversions.
2. The sequence number field is set to 'ØØØ'.
3. If the hour is specified as 'UN' a value of zero is stored.
4. The date of update is set to zeroes.
5. Numeric fields that contain all blanks are copied as zeroes. All other numeric fields must contain leading zeroes.

<u>Remark Record to Card</u> - When a remark record is converted to data card format, the following steps are taken:

1. All corresponding fields are copied with any necessary data conversions.
2. If the hour is zero it is copied as 'UN'.
3. All decimal fields that contain zeroes are copied as blanks. All decimal fields that contain non-zeroes are copied with leading zeroes.

CHAPTER 11

TABLES

File Description - HIS.TABLES

The various HIS tables are stored in the partitioned data set (or library) HIS.TABLES. Each member of the library is a table of data that can be retrieved and updated easily. Each member has a logical record length of 8Ø characters.

Software that accesses the library falls into several categories:

1. General-purpose software. This category includes those programs that can be used to access a variety of tables. Included in this category are two sub-categories:

   a. General-purpose maintenance software - mainline programs that are executed by submitting a HIS command (eg., LISTPDS).

   b. General-purpose access subroutines - subroutines that can be called by other programs (eg., TABLRD).

2. Special-purpose software. This category includes those programs that can only be used to access a single table or a single set of related tables. Included in this category are two sub-categories:

   a. Special-purpose maintenance software - mainline programs that are executed by submitting a HIS command (eg., UPDATE-PROGRAM-TABLE).

   b. Special-purpose access subroutines - subroutines that can be called by other programs (eg., GETCITY).

The LISTPDS Program

LISTPDS can be used to list members of HIS.TABLES. It can also be used to list members of other libraries and to list sequential files. However, it can only be used with libraries and files that meet these constraints:

Library - Record length of 8Ø and block size of 4ØØ.

Sequential file - Record length of 8Ø.

To use LISTPDS, submit a HIS job step and include a command that specifies the program LISTPDS. No parameters are needed on the command. Include a DD statement LISTPDS to enter list options (card formats are described below in this section). Include a DD statement for any library or file being listed other than HIS.TABLES.

The first list option that can be specified is the page-eject card. This card may have either of these formats:

$PAGE-EJECT=NO

$PAGE-EJECT=YES

If no page-eject card is included, no page-eject is used by default. Page-eject specifies that each member or file printed will begin on a new page of output.

To list a sequential file, include a seqname card:

$SEQNAME=ddname

"ddname" is the name of the DD statement included for the file.

To list one or more members of a library, include a library card and one or more member cards. The library card may have either of these formats:

$LIBRARY=ddname

$LIBRARY=TABLES

The first format is used for any library other than HIS.TABLES, and specifies the name of the DD statement included for the library. The second format is used for the HIS.TABLES library (no DD statement has to be included). The member cards have this format:

Columns	Length	Format	Contents
1-8	8	char(8)	Member name 1
9-16	8	char(8)	Member name 2
17-24	8	char(8)	Member name 3
25-32	8	char(8)	Member name 4
33-4Ø	8	char(8)	Member name 5
41-48	8	char(8)	Member name 6
49-56	8	char(8)	Member name 7
57-64	8	char(8)	Member name 8
65-72	8	char(8)	Member name 9
73-8Ø	8	char(8)	Member name 1Ø

Any combination of the member name fields can be coded or left blank. Any number of member name cards can be included after a library card.

The following sample job setup illustrates the use of LISTPDS. This run lists the members LSTMEMA and LSTMEMB of the library SYS1.RUNLIB, the sequential file HIS.PANTBL, and the members LSTMEM1 and LSTMEM2 of HIS.TABLES.

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS
//RUNLIB DD DISP=SHR,DSNAME=SYS1.RUNLIB
//PANTBL DD DISP=SHR,DSNAME=HIS.PANTBL
//SYSIN DD *
:LISTPDS
/*
//LISTPDS DD *
$LIBRARY=RUNLIB
LSTMEMA LSTMEMB
$SEQNAME=PANTBL
$LIBRARY=TABLES
LSTMEM1
LSTMEM2
/*
```

The LOADPDS Program

LOADPDS program can be used to load data into HIS.TABLES. It can also be used to load data into other libraries and into sequential files provided that these constraints are met:

Library - Record length of 8∅ and block size of 4∅∅.

Sequential file - Record length of 8∅.

To use LOADPDS, submit a HIS job step and include a command that specifies the program LOADPDS. No parameters are needed on the command. Include a DD statement named LOADPDS to enter load options and data. Include a DD statement for any library or file being loaded other than HIS.TABLES.

The program can supply sequence numbers (in increments of 1∅) in columns 73 through 8∅ of the output records. A seq card can be used to indicate whether this function is desired:

```
$SEQ=YES
$SEQ=NO
```

$SEQ=YES is the default if this option is not specified.

The program can print a listing of the loaded data. A list card can be used to indicate this function:

```
$LIST=YES
$LIST=NO
```

$LIST=YES is the default for this option.

To load data into a sequential file, include a seqname card:

$SEQNAME=ddname

"ddname" is the name of the DD statement included for the file.

To load data into a library, include a library card and one or more member cards. The library card has one of these formats:

$LIBRARY=ddname
$LIBRARY=TABLES

Use the first format for libraries other than HIS.TABLES (ddname is the name of the DD statement included for the library). Use the second format for HIS.TABLES (no DD statement has to be included). If loading data into HIS.TABLES, specify the parameter DISP=OLD on the EXEC statement. The member card has the following format:

$MEMBER=member-name

Any number of member cards can follow a library card.

Input data is placed immediately following the seqname or member card. The data cards can be in any format except that they cannot contain a dollar sign in column 1.

An eject card can be placed anywhere in the job setup to force a jump to a new page on the printer. The format of this card is:

$EJECT

The following job setup illustrates the use of LOADPDS to load data into HIS.TABLES:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS,DISP=OLD
//SYSIN DD *
:LOADPDS
/*
//LOADPDS DD *
$LIBRARY=TABLES
$MEMBER=LOADA

    Data cards

$MEMBER=LOADB

    Data cards

/*
```

LOADPDS can add a new member or replace an existing member.

<u>The UPDPDS Subroutine</u>

UPDPDS can be used to update a member of HIS.TABLES. It can also be used to update members of other libraries and to update a sequential file provided that these constraints are met:

Library - Record length of 8∅ and block size of 4∅∅.

Sequential file - Record length of 8∅.

To use UPDPDS, submit a HIS job step and include a command that specifies the program UPDPDS. No parameters are needed on the command. If updating HIS.TABLES, specify DISP=OLD on the EXEC statement. Include a DD statement named UPDPDS to enter update options and data. Include a DD statement for any library or file other than HIS.TABLES.

The program can store sequence numbers (in increments of 1∅) in columns 73 through 8∅ of the output records. A seq card can be used to indicate whether these numbers are desired:

```
$SEQ=YES   (default)
$SEQ=NO
```

The program can print a listing of the updated members and file. A list card can be used to indicate whether a listing is desired:

```
$LIST=YES   (default)
$LIST=NO
```

To update a sequential file, include a seqname card:

```
$SEQNAME=ddname
```

"ddname" is the name of the DD statement that defines the file.

To update a library, include a library card and one or more member cards. The library card has one of these formats:

```
$LIBRARY=ddname
$LIBRARY=TABLES
```

Use the first format for libraries other than HIS.TABLES and the second format for HIS.TABLES. "ddname" is the name of the DD statement that defines the library. No DD statement has to be included to update HIS.TABLES.

Update control cards and update data cards are placed immediately after the seqname or library card. Update control cards are identical to the ++C/--C cards of the PANVALET library system. Records can be inserted, deleted, and replaced. To insert records, use a ++C or --C card that specifies the line number after which the records are to be inserted as in these examples:

```
++C 36     (add one or more records after line number 36)
--C Ø      (add one or more records at the beginning of the file)
```

To delete records, specify the range of records on a ++C or --C card:

```
++C 42,48   (delete lines 42 through 48)
--C 22,22   (delete line 22)
```

To replace records, use a delete format card to delete the old lines and place the new records after the delete card. It is not necessary to include the same number of cards as the number of records deleted.

Update data cards can be in any format except that they cannot start with ++, --, or $.

The following job setup is a sample setup for using UPDPDS:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS,DISP=OLD
//SYSIN DD *
:UPDPDS
/*
//UPDPDS DD *
$LIBRARY=TABLES
$MEMBER=MEM1
--C 4
  One or more data cards
--C 8,12
  Zero or more data cards
      .
      .
      .
$MEMBER=MEM2
      .
      .
      .
/*
```

The updates must be submitted in order by line numbers.

<u>The MODPDS Program</u>

MODPDS can delete and rename members of HIS.TABLES. It can also delete and rename members of other libraries. No constraints are placed on record length or record format of libraries.

To use MODPDS, submit a HIS step that contains a command that specifies the MODPDS program. No parameters are needed on the command. If HIS.TABLES is being modified, specify DISP=OLD on the EXEC statement. Include a MODPDS DD statement to indicate operations to be performed. Include a DD statement for any library other than HIS.TABLES being modified.

The first card must be a library card to indicate what library is being modified. The format of this card is:

```
$LIBRARY=ddname
$LIBRARY=TABLES
```

"ddname" is the name of the DD statement that defines the library. No DD statement has to be provided for HIS.TABLES (the second library card format).

The card format for deleting members is:

Columns	Length	Format	Contents
1-8	8	char(8)	Member name 1
9-16	8	char(8)	Member name 2
17-24	8	char(8)	Member name 3
25-32	8	char(8)	Member name 4
33-4Ø	8	char(8)	Member name 5
41-48	8	char(8)	Member name 6
49-56	8	char(8)	Member name 7
57-64	8	char(8)	Member name 8
65-72	8	char(8)	Member name 9
73-8Ø	8	char(8)	Member name 1Ø

Any combination of the member name fields can be coded or left blank. Each of the specified members is deleted.

The card format for renaming a member is:

Columns	Length	Format	Contents
1-3	3	char(3)	$RE
4	1	char(1)	blank
5-12	8	char(8)	Existing name
13	1	char(1)	blank
14-21	8	char(8)	New name
22-8Ø	59	char(59)	blank

The following sample job setup illustrates the use of MODPDS:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS
//XLIB DD DISP=OLD,DSNAME=XLIB
//SYSIN DD *
:MODPDS
/*
//MODPDS DD *
$LIBRARY=XLIB
DELETE1 DELETE2 DELETE#
DELETEG
$RE OLDNAME  NEWNAME
DELETEM
/*
```

This run will delete members DELETE1, DELETE2, DELETE#, DELETEG, and DELETEM, and will rename the member OLDNAME to the name NEWNAME.

The BPAMRD/TABLRD Subroutine

BPAMRD/TABLRD can be used to read members of HIS.TABLES or other libraries. It can only read libraries that have a record length of 8Ø and a block size of 4ØØ. It has these entry points:

TABLRDI	Initialize - open file and locate member
TABLRD	Read a record
TABLRDC	Close the file
BPAMRDI	Initialize - open file and locate member
BPAMRD	Read a record
BPAMRDC	Close the file

Use the TABLRD entry points to read HIS.TABLES. Use the BPAMRD entry points to read other libraries. The subroutine can only be open to one file at any one time, but the file does not have to be closed to switch to another member of the same library.

TABLRDI Entry Point - Call TABLRDI to open the file and locate a member. Pass the char(8) member name. If the member does not exist, the subroutine prints a message and gives an end of file return during the first call to TABLRD.

TABLRD Entry Point - Call TABLRD to read a record. Pass a pointer variable. The subroutine reads a record and returns its address in the pointer. At end of file, a null value is returned.

TABLRDC Entry Point - Call TABLRDC to close the file. No arguments are passed.

BPAMRDI Entry Point - Call BPAMRDI to open the file and locate a member. Pass the char(8) member name and the char(8) name of the DD statement that defines the library. If the member does not exist, the subroutine prints a message and gives an end of file return during the first call to BPAMRD.

BPAMRD Entry Point - Call BPAMRD to read a record. Pass a pointer variable. The subroutine reads a record and returns its address in the pointer. At end of file, a null value is returned.

BPAMRDC Entry Point - Call BPAMRDC to close the file. No arguments are passed.

Error Messages - If an error is detected, an appropriate message is printed. In most cases, a user-1ØØ abend is issued.

> ***** INVALID MEMBER NAME PASSED TO BPAMRD - MEMBER=member, LIBRARY=library
>
> The member name does not start with the characters A-Z.
>
> ***** INVALID LIBRARY NAME PASSED TO BPAMRD - MEMBER=member, LIBRARY=library
>
> The library name does not start with the characters A-Z.
>
> ***** WARNING - MEMBER PASSED TO BPAMRD DOES NOT EXIST - MEMBER=member,LIBRARY=library
>
> Execution continues, but an end of file return is given during the first read call.

The BPAMWR/TABLWR Subroutine

BPAMWR/TABLWR can be used to load data into a member of HIS.TABLES or of another library. It can only process libraries that have a record length of 8Ø and a block size of 4ØØ. It has these entry points:

TABLWRI	Initialization - open file and specify member name
TABLWR	Write a record
TABLWRC	Close the file
BPAMWRI	Initialization - open file and specify member name
BPAMWR	Write a record
BPAMRDC	Close the file

Use the TABLWR entry points to write to HIS.TABLES. Use the BPAMWR entry points to write to other libraries. The subroutine can only be open to one file at any one time, but the file does not have to be closed to switch to another member of the same library.

TABLWRI Entry Point - Call TABLWRI to initialize for writing a member. Pass the char(8) member name. The name can be the name of an existing member (the member will be replaced in its entirety) or the name of a new member.

TABLWR Entry Point - Call TABLWR to write a record. Pass a pointer that contains the record's address.

TABLWRC Entry Point - Call TABLWRC to close the file. No arguments are passed.

BPAMWRI Entry Point - Call BPAMWRI to initialize for writing. Pass the char(8) member name and the char(8) name of the DD statement that defines the library. The member name can be the name of an existing member or a new name.

BPAMWR Entry Point - Call BPAMWR to write a record. Pass a pointer that contains the record's address.

BPAMWRC Entry Point - Call BPAMWRC to close the file. No arguments are passed.

Error Messages - If an error is detected, an appropriate message is printed. In most cases, a user-1ØØ abend is issued.

***** BPAMWR OR TABLWR IS NOT INITIALIZED

The calling program called BPAMWR or TABLWR without opening the file.

***** INVALID MEMBER NAME PASSED TO BPAMWR OR TABLWR - MEMBER=member, LIBRARY=library

The member name does not start with A-Z.

***** INVALID LIBRARY NAME PASSED TO BPAMWRI - MEMBER=member, LIBRARY=library

The library name does not start with A-Z.

***** BPAMWR OR TABLWR - STOW RETURN CODE = n

A bad return code was returned from the STOW macro. The member was not saved.

***** BPAMWR OR TABLWR - LIBRARY BLOCKSIZE IS TOO LARGE

The blocksize is larger than 4ØØ.

The PDSDIR Subroutine

PDSDIR can be called to read the directory entries of a library. The entries are returned in alphabetical order. The format of a directory entry is:

Columns	Length	Format	Contents
1-8	8	char(8)	Member or alias name
9-11	3	bit(24)	TTR of member or alias
12	1	bit(8)	Indicators: 1....... Name is an alias Ø....... Name is not an alias .xx..... Number of user TTR's ...xxxxx Number of halfword in user data field
13-78	Ø-78	varies	User data field

If the members are being read while the directory is being read, two DD statements must be supplied - one for the directory reads and one for the member reads.

PDSDIR has these entry points:

PDSDIRO	Initialization - open the file
PDSDIR	Read a directory entry
PDSDIRC	Close the file

PDSDIRO Entry Point - Call PDSDIRO to open the file. Pass the char(8) DD statement name and a binary fixed(15) return code. Return codes are:

Ø	Successful
1	Unsuccessful - Invalid ddname passed or missing DD statement
2	Unsuccessful - Open failed
3	Unsuccessful - File was already open

PDSDIR Entry Point - Call PDSDIR to read an entry. Pass a pointer variable and a binary fixed(15) return code. Return codes are:

Ø	Successful
1	End of file
2	Unsuccessful - No previous PDSDIRO call
3	Unsuccessful - Called after end of file
4	Unsuccessful - I/O error

If the return code is zero, the pointer contains the address of a directory entry.

PDSDIRC Entry Point - Call PDSDIRC to close the file. Pass a binary fixed(15) return code. Return codes are:

Ø Successful
1 Unsuccessful - file was not open
2 Unsuccessful - close failed

The PDSRD Subroutine

PDSRD can be called to read a member of a partitioned data set. It has these entry points:

PDSRDO	Open the file
PDSRDF	Locate a member
PDSRD	Read a record
PDSRDC	Close the file

PDSRDO Entry Point - Call PDSRDO to open the file. Once the file is open, any number of members can be read before closing the file. Pass the char(8) DD statement name and two binary fixed(15) variables. If the open is successful, the record length of the library is returned in the first binary variable. The second binary variable is a return code:

Ø Successful
1 Unsuccessful - Invalid ddname passed or missing DD statement
2 Unsuccessful - Open failed
3 Unsuccessful - File was already open

PDSRDF Entry Point - Call PDSRDF to locate a member. Pass a binary fixed(15) option code, a pointer, and a binary fixed(15) return code. The option code must have one of these values:

Ø Member name is passed
1 Directory entry is passed

The pointer must contain the address of a member name or of a directory entry. If a member name is passed, the name can be passed in a char(8) variable rather than through a pointer. Return codes are:

Ø Successful
1 Unsuccessful - Name not found
2 Unsuccessful - I/O error or invalid TTR in directory entry
3 Unsuccessful - No previous PDSRDO call
4 Unsuccessful - Option code neither Ø nor 1

PDSRD Entry Point - Call PDSRD to read a record. Pass a pointer variable and a binary fixed(15) return code variable. Return codes are:

- Ø Successful
- 1 End of file
- 2 Unsuccessful - No previous successful PDSRDF call
- 3 Unsuccessful - Called after end of file
- 4 Unsuccessful - I/O error

PDSRDC Entry Point - Call PDSRDC to close the file. Pass a binary fixed(15) return code variable. Return codes are:

- Ø Successful
- 1 Unsuccessful - File was not open
- 2 Unsuccessful - Close failed

The LIST-PDS-DIREC Program

LIST-PDS-DIREC can list the directory of a library. To use, submit a HIS job step and include a command that specifies the LIST-PDS-DIREC program. Code the parameter DDNAME=ddname on the command to specify the name of the DD statement that defines the library. For HIS.TABLES, specify DDNAME=TABLES. A sample job setup is:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS
//CRJELIB DD DISP=SHR,DSNAME=CRJE.LIB.user-id
//SYSIN DD *
:LIST-PDS-DIREC,DDNAME=TABLES
:LIST-PDS-DIREC,DDNAME=CRJELIB
/*
```

A LOCATION parameter may be added to the command to list only a portion of the directory. This parameter can take on two formats:

```
LOCATION=name    Print names that begin with "name"
LOCATION=name1-name2    Print names between the two names, inclusive
```

Examples:

```
LOCATION=XYZ            Print names that start with XYZ
LOCATION=A-J9999999     Print names that start with A-J
```

The LIST-PDS-MEMBERS Program

LIST-PDS-MEMBERS prints the contents of members of libraries. The members are printed in alphabetical order.

To use, submit a HIS job step with a LIST-PDS-MEMBERS command. If listing a library other than HIS.TABLES, include a DD statement for the library. Command parameters are:

DDNAME=ddname (required) Code the name of the DD statement of the library. Code TABLES for HIS.TABLES

LOCATION=name (optional) Only those members whose name begins with the specified name are printed. Default is entire library.

LOCATION=name1-name2 (optional) Prints those members whose names lie between the specified names. Default is entire library.

LIST=72/8Ø/CRJE-88/RECSIZE (optional) Specifies list format.

- LIST=72 — Print only first 72 characters of record.
- LIST=8Ø — Print only first 8Ø characters of record.
- LIST=CRJE-88 — (For CRJE libraries) Characters 1-8 are the sequence numbers and characters 9-88 are the record.
- LIST=RECSIZE — Print entire record to maximum of 122 characters.

Defaults:

Record length of 88 - CRJE-88

Other record lengths - RECSIZE

A sample job setup is:

```
// JOB    (a JOB statement is obtained from the Data Processing Bureau)
// EXEC HIS
//CRJELIB DD DISP=SHR,DSNAME=CRJE.LIB.user-id
//SYSIN DD *
:LIST-PDS-MEMBERS,DDNAME=CRJELIB,LOCATION=ABC
:LIST-PDS-MEMBERS,DDNAME=TABLES,LOCATION=PROGTBL
/*
```

The Program Name Table

The record format of the program table is:

Columns	Length	Format	Data Element
1-2Ø	2Ø	char(2Ø)	External program name
21-28	8	char(8)	Internal program name
29-8Ø	52		Unused at this time

The external program name is the name coded on a command (eg., LIST-&-SUM). The internal program name is the load module name (eg., HIS3Ø2ØØ).

The LIST-PROGRAM-TABLE Program - LIST-PROGRAM-TABLE prints the contents of the program name table. To use, submit the following:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS
//SYSIN DD *
:LIST-PROGRAM-TABLE
/*
```

The UPDATE-PROGRAM-TABLE Program - UPDATE-PROGRAM-TABLE updates the program name table. Records can be added, revised, and deleted. The card format of update cards is:

Columns	Length	Data Element
1-8	8	Internal program name
9	1	Blank
1Ø-29	2Ø	External program name
3Ø-8Ø	51	Blank

To delete a record, code either the internal or the external name and leave the other name blank. To add a record, code both fields (neither name can already be stored in the table). To change an external name, code the internal name as stored and the new external name. To change an internal name, code the external name as stored and the new internal name. Submit the cards with the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS,DISP=OLD
//SYSIN DD *
:UPDATE-PROGRAM-TABLE,DDNAME=UPDPROG,PASSWORD=password
/*
//UPDPROG DD *
    Data cards in any order
/*
```

The Passed Parameter Table

The record format of the passed parameter table is:

Columns	Length	Format	Data Element
1-8	8	char(8)	Load module name
9-1Ø	2		Unused at this time
11-72	62	char(62)	Parameter to be passed
73-8Ø	8		Unused at this time

If an entry is stored in this table for a load module, the parameter is passed to the load module whenever it is executed unless an overriding PARM parameter is coded on the command. If no entry is stored for a load module and no PARM parameter is coded on the command, the parameter REPORT,ISASIZE(2ØK) is passed.

The PASSPARM-TABLE Program - PASSPARM-TABLE can be used to list, update, or load the passed parameter table. To list the table, submit the following job setup:

```
// JOB    (a JOB card is obtained from the data processing bureau)
// EXEC HIS
//SYSIN DD *
:PASSPARM-TABLE,TYPE-RUN=LIST
/*
          TYPE-RUN=SORT-&-LIST can be specified instead of
             TYPE-RUN=LIST
```

To update the table, prepare data cards in the same format as the record format of the table. To delete a record, code only the load module name. To insert or rewrite, code both fields. Submit the cards with the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS,DISP=OLD
//SYSIN DD *
:PASSPARM-TABLE,TYPE-RUN=UPDATE-&-LIST,DDNAME=UPDPARM
/*
//UPDPARM DD *
   Data cards in any order
/*
```

To load the file, code data cards in the format of the record. Submit these cards with the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS,DISP=OLD
//SYSIN DD *
:PASSPARM-TABLE,TYPE-RUN=BUILD-&-LIST,DDNAME=BLDPARM
/*
//BLDPARM DD *
   Data cards in any order
/*
```

The Instruction Parameter Table

The record format of the Instruction Parameter table is:

Columns	Length	Format	Data Element
1-1∅	1∅	char(1∅)	Keyword name
11	1	char(1)	Parameter format
12	1	char(1)	Instruction segment
13-14	2	fixed(2)	Field length in bytes
15-16	2	fixed(3)	Position within instruction segment
17-8∅	64		Unused at this time

The keyword name is the name coded in the command for the parameter (eg., DATA in DATA=PRIMARY). The parameter format is:

C   Character format
P   Packed decimal format

The instruction segment is:

S   System segment
P   Program segment

The field length is specified in bytes. The position of the field is the byte number with the instruction segment (the first byte of a segment is ØØ1).

The LIST-PARM-TABLE Program - LIST-PARM-TABLE prints the contents of the table. Submit the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS
//SYSIN DD *
:LIST-PARM-TABLE
/*
```

The UPDATE-PARM-TABLE Program - UPDATE-PARM-TABLE updates the table. Prepare data cards in this format:

Columns	Length	Data element
1-1Ø	1Ø	Keyword name
11	1	Parameter format
12	1	Instruction segment
13-14	2	Field length (code leading zeroes)
15-17	3	Position in segment (code leading zeroes)
18-8Ø	63	Blank

To delete a record, code only the keyword name. To add or rewrite a record, code all fields. Submit the cards with the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS,DISP=OLD
//SYSIN DD *
:UPDATE-PARM-TABLE,DDNAME=UPDPARM,PASSWORD=password
/*
//UPDPARM DD *
   Data cards in any order
/*
```

The Equivalence Table

The record format of the equivalence table is:

Columns	Length	Format	Data Element
1-8	8	char(8)	Access name
9-16	8	char(8)	Load module name
17-8Ø	64		Unused at this time

The equivalence table provides the load module names of dynamic subroutines. The interface subroutine requests a dynamic subroutine by specifying an access name (eg., RLGRDQ). The equivalence table provides the corresponding load module name (eg., HIS3ØØØØ). The equivalence table can be overridden (in part or in whole) by including an EQUIV DD statement with a run:

```
        :  (normal job setup)
      //EQUIV DD *
        Data cards in record format
      /*
```

The EQUIV-TABLE Program - EQUIV-TABLE can be used to list, update, or load the equivalence table. To list the table, submit the following job setup:

```
// JOB   (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS
//SYSIN DD *
:EQUIV-TABLE,TYPE-RUN=LIST
/*
      (TYPE-RUN=SORT-&-LIST may be specified)
```

To update the table, prepare data cards in the same format as the record format of the table. To delete a record, code only the access name. To change the load module name in a record, code the access name as stored and the new load module name. To insert a record, code the access name (which cannot already be stored) and the load module name. Submit the cards with this job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS,DISP=OLD
//SYSIN DD *
:EQUIV-TABLE,TYPE-RUN=UPDATE-&-LIST,DDNAME=UPDEQUIV
/*
//UPDEQUIV DD *
   Data cards in any order
/*
```

To load the table, prepare data cards in the same format as the record format of the table. Submit the cards with this job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS,DISP=OLD
//SYSIN DD *
:EQUIV-TABLE,TYPE-RUN=BUILD-&-LIST,DDNAME=BLDEQUIV
/*
//BLDEQUIV DD *
   Data cards in any order
/*
```

The City Table

The record format of the city table is:

Columns	Length	Format	Data Element
1-2	2	fixed(3)	City number
3-2Ø	18	char(18)	City name - left-justified
21-38	18	char(18)	City name - centered
39-56	18	char(18)	City name - left-justified with internal blanks replaced with hyphens
57	1	fixed(1)	Population code
58-59	2	fixed(2)	County number - alphabetical system
6Ø-61	2	fixed(2)	County number - registration system
62-65	4	char(4)	Federal city number
66-72	7	char(7)	Population
73-8Ø	8		Unused at this time

The population code has these values:

1	Ø-999
2	1,ØØØ-2,499
3	2,5ØØ-4,999
4	5,ØØØ-9,999
5	1Ø,ØØØ-24,999
6	25,ØØØ-49,999
7	5Ø,ØØØ and over

The LIST-CITY-TABLE Program - LIST-CITY-TABLE prints the contents of the city table. To use, submit the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS
//SYSIN DD *
:LIST-CITY-TABLE
/*
```

The UPDATE-CITY-TABLE Program - UPDATE-CITY-TABLE can rewrite records in the city table. It cannot add or delete records. The data card format of update cards is:

Columns	Length	Contents
1-3	3	City number
4-21	18	City name - left-justified
22-39	18	City name - centered
4Ø-57	18	City name - left-justified with hyphens
58	1	Population code
59-6Ø	2	County number - alphabetical system
61-62	2	County number - registration system
63-66	4	Federal city number
67-73	7	Population
74-8Ø	7	Blank

Code the city number and any fields being changed. Submit the cards with the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS,DISP=OLD
//SYSIN DD *
:UPDATE-CITY-TABLE,DDNAME=UPDCITY,PASSWORD=password
/*
//UPDCITY DD *
   Data cards in order by city number
/*
```

The CVTCITY Subroutine - CVTCITY accesses the city table to determine the city number of a passed city name. Pass the char(18) city name and a fixed(3) variable. If the name is a valid name found in the table, the city number is returned. If all blanks are passed, zero is returned. If the name is not found in the table, a value of -1 is returned. The name can be passed in either left-justified format or in left-justified with hyphens format.

The INCITY Subroutine - INCITY is a subroutine for accessing data from the city table. This subroutine is implicitly dependant upon a table that contains 126 entries with numbers ranging from 1 to 126, and for this reason its use is discouraged in all new programs. When called, it reads data elements from the table and stores them in arrays provided by the calling program. The arrays have bounds (Ø:126). All blanks or zeroes are stored in the zeroeth element. The calling program passes a binary fixed(15) option code

and a pointer. The option code indicates which data element(s) are needed. The code is calculated by adding together the numbers that correspond to the data elements in this list:

1	City name - left-justified
2	City name - centered
4	City name - left-justified with hyphens
8	Population code
16	County number - alphabetical
32	County number - registration
64	Federal city number
128	Population

The pointer has the address of the array(s). If more than one data element is requested, the arrays must be contiguous in core and in the above order. This is accomplished by placing the arrays within a structure.

The GETCITY Subroutine - GETCITY is a subroutine for reading the city table. This subroutine is independant of table size. It has these entry points:

GETCITI	Initialization - read table into core
GETCITY	Return a record
GETCITC	Free core used for table

Records returned by GETCITY are in slightly different format than those stored in the table. The record as returned has this format:

Columns	Length	Format	Data Element
1-3	3	fixed(5)	City number
4-21	18	char(18)	City name, left-justified
22-39	18	char(18)	City name, centered
4Ø	1	fixed(1)	Population code
41-42	2	fixed(3)	County number - alphabetical
43-44	2	fixed(3)	County number - registration
45-48	4	char(4)	Federal city number
49-55	7	char(7)	Population

GETCITI Entry Point - Call GETCITI to read the table into core for future access. No arguments are passed.

GETCITY Entry Point - Call GETCITY to retrieve a record. Pass the fixed(5) city number and a pointer. If the passed city number corresponds to a

a record in the table, the record's address is returned in the pointer. Otherwise, a null pointer is returned.

GETCITC Entry Point - Call GETCITC to free the core used for holding the city table. No arguments are passed.

The CNTCITY Subroutine - CNTCITY returns a count of the number of records in the city table. Pass a fixed(5) variable.

The County Table

The record format of the county table is:

Columns	Length	Format	Data Element
1-2	2	fixed(2)	County number - alphabetical system
3-4	2	fixed(2)	County number - registration system
5-6	2	fixed(2)	Financial district
7-21	15	char(15)	County name - left-justified
22-36	15	char(15)	County name - centered
37-51	15	char(15)	County name - left-justified with internal blanks replaced with hyphens
52	1	fixed(1)	Patrol division
53	1	char(1)	EMS division
54-8Ø	27	char(27)	Unused

The LIST-COUNTY-TABLE Program - LIST-COUNTY-TABLE prints the contents of the county table. To use, submit the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS
//SYSIN DD *
:LIST-COUNTY-TABLE
/*
```

The UPDATE-COUNTY-TABLE Program - UPDATE-COUNTY-TABLE can rewrite records in the county table. It cannot add or delete records. The data card format of update cards is:

Columns	Length	Data Element
1-2	2	County number - alphabetical system
3-4	2	County number - registration system
5-6	2	Financial district
7-21	15	County name - left-justified
22-36	15	County name - centered
37-51	15	County name - left-justified with internal hyphens
52	1	Patrol division
53	1	EMS division
54-8∅	27	Blank

Code the alphabetical county number and any fields being changed. Submit the cards with the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS,DISP=OLD
//SYSIN DD *
:UPDATE-COUNTY-TABLE,DDNAME=UPDCNTY,PASSWORD=password
/*
//UPDCNTY DD *
   Data cards in order by alphabetical county number
/*
```

The CVTCNTY Subroutine - CVTCNTY converts a county name to the equivalent county number. Pass the char(15) county name (left-justified or left-justified with hyphens) and two fixed(3) variables. The alphabetical number is returned in the first fixed(3) variable and the registration number in the second. A value of zero is returned if all blanks are passed. A value of -1 is returned if the passed name is not found in the table.

The INCNTY Subroutine - INCNTY is a subroutine for accessing the county table. It is implicitly dependant upon a table with 56 records, and for this reason its use is discouraged in all new programs. When called, it reads data elements from the table and stores them in arrays passed by the calling program. The arrays have bounds (1:56). The calling program passes a binary fixed(15) index code, a binary fixed(15) option code, and a pointer. The index code specifies which number system is used as an index in the arrays

∅ Alphabetical
1 Registration

The option code indicates which data elements are to be read. Compute the value by adding the appropriate numbers in the following list:

1	County name - left-justified
2	County name - centered
4	County name - left-justified with hyphens
8	County number of opposite system from index
16	Financial districts
32	Patrol division

The pointer contains the address of the array(s). If two or more arrays are being filled, they must be contiguous in core and in the above order. This is accomplished by placing the arrays in a structure.

The GETCNTY Subroutine - GETCNTY is a subroutine for accessing the county table. It is independant upon table size. It has these entry points:

GETCNTI	Initialization - read the table into core
GETCNTY	Return a record
GETCNTC	Free the core used by the table

GETCNTI Entry Point - Call GETCNTI to read the table into core. No arguments are passed.

GETCNTY Entry Point - Call GETCNTY to access a record. Pass a binary fixed(15) numbering system, a fixed(3) county number, and a pointer. The binary variable must contain one of these values:

Ø	Alphabetical
1	Registration

If a record is stored for the passed number, its address is returned in the pointer. Otherwise, a null pointer is returned.

GETCNTC Entry Point - Call GETCNTC to free the core needed for the table. No arguments are passed.

The CNTCNTY Subroutine - CNTCNTY returns a count of the number of records in the county table. Pass a fixed(3) variable.

The Project Class Table

The record format of the project class table is:

Columns	Length	Format	Data Element
1-2	2	fixed(2)	Administration code
3-6	4	char(4)	Project classification
7-8Ø	74		Unused at this time

The project class table equates administration codes with project classifications. Both of these items are coded in roadlog data cards but the administration code is not stored because it can be derived from the project classification.

The LIST-PROJECT-TABLE Program - LIST-PROJECT-TABLE prints the contents of the project class table. To use, submit the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS
//SYSIN DD *
:LIST-PROJECT-TABLE
/*
```

The UPDATE-PROJECT-TABLE Program - UPDATE-PROJECT-TABLE can delete, insert, and rewrite records in the project class table. Prepare data cards in the same format as the project class record. To delete a record, code only the administration code. To rewrite a record, code its administration code and the new project classification. To insert a record, code both fields (the administration code must not equal that of an existing record or the existing record is rewritten). Submit the data cards with the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS,DISP=OLD
//SYSIN DD *
:UPDATE-PROJECT-TABLE,DDNAME=UPDPROJ,PASSWORD=password
/*
//UPDPROJ DD *
   Data cards in any order
/*
```

The CVTPROJ Subroutine - CVTPROJ can convert a project classification to its equivalent administration code and vice versa. Pass a char(11) variable and a fixed(2) variable. To convert a project class to an administration code:

Pass the project class or the complete project number left-justified in the char(11) variable.

Pass a value of zero in the fixed(2) variable.

The administration code is returned in the fixed(2) variable. A value of -1 is returned if the project class passed is not found in the table.

If a non-zero value is passed to CVTPROJ in the fixed(2) variable, the char(11) variable will contain *ERR and the fixed(2) variable will contain -1 upon return.

To convert an administration code to a project class:

Pass all blanks in the char(11) variable.

Pass the administration code in the fixed(2) variable.

The project class is returned left-justified in the char(11) variable. The string *ERR is returned if the administration code is not found in the table.

If a non-blank value is passed in the char(11) variable, the string *ERR is returned in the char(11) variable and a value of -1 is returned in the fixed(2) variable.

If all blanks are passed in the char(11) variable and a value of zero is passed in the fixed(2) variable, no operations are performed (neither variable is modified).

The INPROJ Subroutine - INPROJ reads the project class table into a core array supplied by the calling program. Pass an array of char(4) variables with bounds (Ø:99). The project class table is read into the array using the administration code as the array index. Blanks are stored in any element that corresponds to a number not found in the table.

The Surface Type Table

The record format of the surface type table is:

Columns	Length	Format	Data Element
1-3	3	fixed(4)	Surface type
4	1	fixed(1)	Surface type classification
5-8Ø	76		Unused at this time

The surface type table equates surface types (as coded in the roadlog file) to a 1-digit surface classification. The classifications are shown in this table:

1	Primitive
2	Unimproved
3	Graded and drained
4	Gravel
5	Bituminous surface treatment
6	Road mix
7	Plant mix
8	Portland cement concrete

The LIST-SURFACE-TABLE Program - LIST-SURFACE-TABLE prints the contents of the surface type table. To use, submit the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS
//SYSIN DD *
:LIST-SURFACE-TABLE
/*
```

The UPDATE-SURFACE-TABLE Program - UPDATE-SURFACE-TABLE can delete, insert, and rewrite records in the surface type table. Prepare data cards in this format:

Columns	Length	Data Element
1-4	4	Surface type
5	1	Surface type classification
6-8Ø	75	Blank

To delete a record, code only the surface type. To rewrite or insert, code both fields (if the coded surface type is found in the table, the card is considered to be a rewrite card). Submit the data cards with the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS,DISP=OLD
//SYSIN DD *
:UPDATE-SURFACE-TABLE,DDNAME=UPDSURF,PASSWORD=password
/*
//UPDSURF DD *
   Data cards in any order
/*
```

The CVTSURF Subroutine - CVTSURF returns the classification of a passed surface type. Pass a fixed(4) variable and a fixed(1) variable. Pass the surface type in the fixed(4) variable. The classification is returned in the fixed(1) variable. A value of zero is returned if the passed surface type is not found in the table.

The Sufficiency Table

The record format of the sufficiency table is:

Columns	Length	Format	Data Element
1-5	5	char(5)	Record identifier
6-8	3	fixed(4)	Factor 1
9-11	3	fixed(4)	Factor 2
12-14	3	fixed(4)	Factor 3
15-17	3	fixed(4)	Factor 4
18-2∅	3	fixed(4)	Factor 5
21-23	3	fixed(4)	Factor 6
24-26	3	fixed(4)	Factor 7
27-29	3	fixed(4)	Factor 8
3∅-32	3	fixed(4)	Factor 9
33-35	3	fixed(4)	Factor 1∅
36-38	3	fixed(4)	Factor 11
39-8∅	42		Unused at this time

1∅∅ records are stored in the table. Some have fewer factors than 11 stored - the extra positions are filled with blanks. There are five types of records:

Code	Records	Factors	Name of record
HV	16	11	Hourly volume
AF	6∅	5	Adjustment factors
LF	2∅	3	Lane factors
SF	3	4	Speed factors
WF	1	5	Width factors

The record identifier in columns 1-5 consists of the record type code, a hyphen, and the record number (eg., HV-Ø1 is the first hourly volume record, AF-6Ø is the last adjustment factor record). The factors in the hourly volume records are fixed(4). The factors in the other records are fixed(4,2).

<u>The LIST-SUFF-TABLE Program</u> - LIST-SUFF-TABLE prints the contents of the sufficiency table. To use, submit the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS
//SYSIN DD *
:LIST-SUFF-TABLE
/*
```

<u>The UPDATE-SUFF-TABLE Program</u> - UPDATE-SUFF-TABLE can rewrite records in the table. Prepare data cards in this format:

Columns	Length	Data Element
1-5	5	Record identifier
6-9	4	Factor 1
1Ø-13	4	Factor 2
14-17	4	Factor 3
18-21	4	Factor 4
22-25	4	Factor 5
26-29	4	Factor 6
3Ø-33	4	Factor 7
34-37	4	Factor 8
38-41	4	Factor 9
42-45	4	Factor 1Ø
46-49	4	Factor 11
5Ø-8Ø	31	Blank

Code the record identifier of the record being updated and code any factors that are being changed. Submit the cards with the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS,DISP=OLD
//SYSIN DD *
:UPDATE-SUFF-TABLE,DDNAME=UPDSUFF,PASSWORD=password
/*
//UPDSUFF DD *
   Data cards in any order
/*
```

The GETSUFF Subroutine - GETSUFF reads the sufficiency table into a set of five arrays provided by the calling program. Pass these arguments:

1. HV array - (16,11) fixed(4)
2. AF array - (4,5,15) fixed(4,2)
3. LF array - (4,5,3) fixed(4,2)
4. SF array - (3,4) fixed(4,2)
5. WF array - (5) fixed(4,2)
6. Return code - binary fixed(15)

A return code of zero indicates successful completion. A return code of one indicates an error in the sufficiency table.

The File Rewrite Tables

The record format of file rewrite tables is:

Columns	Length	Format	Data Element
1-2Ø	2Ø	char(2Ø)	Data element name
21	1	char(1)	Record type
22	1	char(1)	Data element format
23-24	2	fixed(3)	Field length
25-26	2	fixed(3)	Digits to right of decimal point
27-28	2	fixed(3)	Offset in record format
29-3Ø	2	fixed(3)	Offset in data card/string format
31-8Ø	5Ø		Unused at this time

The file rewrite tables describe the data elements in data card and record formats. They are used by the rewrite function of the update programs.

Most records contain a blank in the record type field. Other codes used are:

K Field described is the record's key

D Field described is a sub-field of another field

R Field described is stored in record but not in data card

E Field described is stored in data card but not in record

The data element format is:

C Character format

P Packed-decimal format (fixed format)

The field length provides the length of the field. For character formats, specify the length in characters. For decimal formats, specified the length in digits.

The offsets are specified in bytes. The first byte of a record is byte number 1.

The first record stored in a file rewrite table is a header record with this format:

Columns	Length	Format	Data Element
1-2	2	fixed(3)	Size of largest character field
3-4	2	fixed(3)	Size of largest decimal field
5-7	3	fixed(5)	Number of records with blank type
8-1Ø	3	fixed(5)	Number of type D records
11-13	3	fixed(5)	Number of type E records
14-16	3	fixed(5)	Number of type K records
17-19	3	fixed(5)	Number of type R records

The LIST-FILE-TABLE Program - LIST-FILE-TABLE prints the contents of a file rewrite table. Specify the 3-character file name in the FILE-NAME parameter. File names are:

BDG	Bridge file
DAC	Accident detail file
RLG	Roadlog file
RRX	Railroad file
SKD	Skid file
SUF	Sufficiency file
TRF	Traffic file
TRM	True mileage file
USA	Urban sign file - assembly records
USS	Urban sign file - sign records
USR	Urban sign file - remark records
VAC	Accident vehicle file

Use the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS
//SYSIN DD *
:LIST-FILE-TABLE,FILE-NAME=name
/*
```

The BUILD-FILE-TABLE Program - BUILD-FILE-TABLE can be used to load a new file table or to replace an existing table in its entirety. Prepare data cards in this format:

Columns	Length	Data Element
1-4	4	Blank
5-24	2∅	Data element name
25	1	Type code
26	1	Data element format
27-29	3	Field length
3∅-31	2	Number of digits to right of decimal point
32-34	3	Offset in record
35-37	3	Offset in data card/string
38-8∅	43	Blank

The header record is supplied automatically by BUILD-FILE-TABLE and need not be coded. Submit the data cards with this job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS,DISP=OLD
//SYSIN DD *
:BUILD-FILE-TABLE,FILE-NAME=name,DDNAME=BLDFILE,PASSWORD=password
/*
//BLDFILE DD *
   Data cards in order to be stored
/*
```

The UPDATE-FILE-TABLE Program - UPDATE-FILE-TABLE can delete, insert, and rewrite records in a file table. Data cards are coded in the same format as for BUILD-FILE-TABLE. All fields must be coded. Combine these cards with update control cards (++C n, ++C n,n, --C n, --C n,n) that indicate what records are to be deleted and where inserts are to be stored. These control cards are described above under the UPDPDS program. Submit the cards with this job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS,DISP=OLD
//SYSIN DD *
:UPDATE-FILE-TABLE,FILE-NAME=name,DDNAME=UPDFILE,PASSWORD=password
/*
//UPDFILE DD *
   Data cards and update control cards
/*
```

The REWRITE Subroutine - REWRITE performs substitutions into a record from coded updates. It is used by the rewrite functions of update programs. The rewrite function performs these steps:

1. Read a data card.
2. Read the record for update and convert to data card format.
3. Pass the data card and the converted record to REWRITE for substitutions.
4. Edit checks on new record.
5. Convert new record back to data record format.
6. Rewrite the record.

When calling REWRITE, pass a char(3) variable and two pointers. The char(3) variable contains the file-name (eg., RLG). The first pointer contains the address of the record in data card format. The second pointer contains the address of the update card from which substitutions are to be made. All non-blank fields (as described in the file table) are copied into the record.

The REWRITA Entry Point - The file table is kept in core during the execution of the program. If more than one table is needed during the run, call REWRITA and pass the binary fixed(15) number of tables that will be needed.

The REWRITE Entry Point - Call REWRITE to perform substitutions. Pass variables as described above. During the first call for a given name, the file table is read into core and saved for future calls.

The REWRITF Entry Point - Call REWRITF to free storage used for the file table(s). No arguments are passed.

The PTW Table

The record format of the PTW table is:

Columns	Length	Format	Data Element
1-15	15	char(15)	Starting key of PTW section
16	1	char(1)	Direction
17-8∅	64		Unused at this time

The PTW table provides the locations of Present Traveled Way sections. PTWs are sections of primary highway that are scheduled for replacement by interstate highways. The starting milepoint coded in the table is the starting key based on the primary system, and must correspond to the key of a CO record in the roadlog file. The direction code is a blank if the primary highway and the interstate highway are mileposted in the same direction and is an X if the two routes are mileposted in opposite directions.

The LIST-PTW-TABLE Program - LIST-PTW-TABLE prints the contents of the PTW table. It also retrieves the ending primary milepoint and the starting and ending interstate milepoints of each PTW section and prints this information. To use, submit this job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HISRLG
//SYSIN DD *
:LIST-PTW-TABLE
/*
```

Maintaining the PTW Table - To load or update the table, use the programs LOADPDS and UPDPDS described above. LISTPDS can be used to list the table.

The INPTW Subroutine - INPTW reads the PTW table into a core array, including the ending primary milepoint and the starting and ending interstate milepoints from the roadlog file. Pass a binary fixed(15) variable and a pointer. The number of entries is returned in the binary variable. Prior to calling INPTW, set the pointer to the address of a structure for storing the table:

```
1  ENTRY(25),
   2  PRIMARY_START       CHAR(15),
   2  PRIMARY_END         CHAR(15),
   2  INTERSTATE_START    CHAR(15),
   2  INTERSTATE_END      CHAR(15),
   2  DIRECTION           CHAR(1);
```

The TESTPTW Subroutine - TESTPTW checks a milepoint to see if it is a primary milepoint located within a PTW section. Pass the char(15) milepoint, a second char(15) variable, and a binary fixed(15) return code variable. Return codes are:

Ø Milepoint not in PTW section
1 Milepoint in PTW section

If the return code is Ø, the first milepoint is copied into the second char(15) variable. If the return code is 1, the equivalent milepoint specified on the interstate system is returned in the second char(15) variable.

The Password Table

The password table contains the passwords needed to run password-protected programs.

The LIST-PASSWORD-TABLE Program - LIST-PASSWORD-TABLE lists the password table. To use, submit the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS
//SYSIN DD *
:LIST-PASSWORD-TABLE,PASSWORD=password
/*
```

The BUILD-PASSWORD-TABLE Program - BUILD-PASSWORD-TABLE loads the password table. To use, submit the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS
//SYSIN DD *
:UPDATE-PASSWORD-TABLE,DDNAME=BLDPASS,PASSWORD=password
/*
//BLDPASS DD *
   Data cards
/*
```

The Load Module Table

The record format of the load module table is:

Columns	Length	Format	Data Element
1-8	8	char(8)	Load module name
9	1		Blank
1Ø-17	8	char(8)	Entry point name
18	1		Blank
19-26	8	char(8)	Access name (dynamic subroutines only)
27	1		Blank
28-35	8	char(8)	Source module name
36	1		Blank
37-44	8	char(8)	Source module name
45	1		Blank
46-53	8	char(8)	Source module name
54	1		Blank
55-62	8	char(8)	Source module name
63	1		Blank
64-71	8	char(8)	Source module name
72	1		Blank
73-8Ø	8		Unused at this time

The load module table describes the load modules that comprise the software of the Highway Information System. Each load module has one or more records in the table. The entry point name and the access name contain blanks in all but the first record for a load module.

The LIST-LOADMOD-TABLE Program - LIST-LOADMOD-TABLE prints a formatted listing of the load module table. No parameters are required on the command. Optional parameters include:

LIST=WIDE/NARROW - Use wide format for paper width of 14 inches and narrow format for paper width of 8½ inches.

SUBSYSTEM=name - Specifies that only load modules of the specified subsystem are listed. "name" is five characters in length, as in SUBSYSTEM=HIS33 (accident subsystem). Valid subsystems are:

HIS2Ø	Supervisory software
HIS21	Tables software
HIS22	Select subsystem
HIS23	Utilities subsystem
HIS3Ø	Roadlog subsystem
HIS31	Traffic subsystem
HIS32	True mileage subsystem

HIS33	Accident subsystem
HIS34	Sufficiency subsystem
HIS35	Bridge subsystem
HIS36	Railroad subsystem
HIS37	Grid table subsystem
HIS38	Urban sign inventory subsystem
HIS39	Skid subsystem
HIS4Ø	Open range subsystem

To use, submit the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS
//SYSIN DD *
:LIST-LOADMOD-TABLE,...
/*
```

<u>The LIST-SOURCE-XREF Program</u> - The LIST-SOURCE-XREF produces a cross-reference listing of source modules showing all load modules in which each source module is included. The LIST and SUBSYSTEM parameters can be used as in LIST-LOADMOD-TABLE (note that SUBSYSTEM specifies the load modules that are examined and not the source modules). To use, submit the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS
//SYSIN DD *
:LIST-SOURCE-XREF,...
/*
```

<u>Maintaining the Load Module Table</u> - Use the programs LISTPDS, LOADPDS, and UPDPDS described above to maintain the load module table.

<u>The Select Tables</u>

The first record of each select table contains a file I.D. number in columns 1-4 and blanks in the remaining columns. A list of file I.D. numbers is included in the Programming Details manual. The format of the remaining records in each select table is:

Columns	Length	Format	Data Element
1-3	1	char(3)	Option code number
4	1	char(1)	Format code
5-6	2	char(2)	Field length
7-8	2	char(2)	Digits to right of decimal point
9-11	3	char(3)	Offset in record
12-31	2Ø	char(2Ø)	Data element name
32-8Ø	49		Unused at this time

The option code number is present only in records that describe a data element for which special processing is needed. This code is used by the select software to indicate what type of processing is needed. The numeric code is divided into these ranges:

ØØ1-399 Special processing required when select statement is read

4ØØ-599 Special processing required both when select statement is read and when a record is read

6ØØ-999 Special processing required when a record is read

Currently implemented codes are:

ØØ1 Data element contains an alphabetic county number. If a county name is coded in a select statement, it is converted to the equivalent county number.

ØØ2 Data element contains a registration county number. If a county name is coded in a select statement, it is converted to the equivalent county number.

ØØ3 Data element contains a city number. If a city name is coded in a select statement, it is converted to the equivalent city number.

4ØØ Data element contains a project number. If a project class is specified in a select statement, it is converted to an administration code. When a record is read, the administration code is extracted from the project number.

6ØØ Requests select based on population code. When a record is read, the population code is determined from the city number.

6Ø1 Requests select based on financial district. When a record is read, the financial district is determined from the county number.

6Ø2 Requests select based on patrol division. When a record is read, the patrol division is determined from the county number.

6Ø3 Requests select based on surface type class. When a record is read, the surface type class is determined from the surface type.

Format codes are:

- C Character format
- P Packed-decimal (fixed) format
- D Date format

Date format data elements must consist of three 2-digit fields in the order month, day, and year.

The field length specifies the length of the data element in the record. For character format, specify the length in characters. For fixed format, specify the length in digits (if there is a decimal point, specify the number of digits to the right of the decimal point in columns 7-8). For date format, the length is always six.

The offset in record field specifies the starting byte of the record of the data element. The first byte in a record is byte number ØØ1.

<u>The LIST-SELECT-TABLE Program</u> - LIST-SELECT-TABLE prints the contents of a select table. To use, submit the following job setup:

```
// JOB    (a JOB card is obtained from the Data Processing Bureau)
// EXEC HIS
//SYSIN DD *
:LIST-SELECT-TABLE,FILE-NAME=name
/*
```

The file names are chosen from this list:

- ACD Accident directory file
- BDG Bridge file
- DAC Accident detail file
- RLG Roadlog file
- RRX Railroad file
- SKD Skid file
- SUF Sufficiency file

TRF	Traffic file
USN	Urban sign inventory file
VAC	Accident vehicle file

Maintaining the Select Tables - Use the programs LISTPDS, LOADPDS, and UPDPDS to maintain the tables. These programs are described earlier in this chapter.

Field-List Tables

Field-list tables are used by programs that implement the FIELD-LIST parameter. The tables contain a set of records that is printed whenever FIELD-LIST=YES is specified on the command. The tables are maintained by the LISTPDS, LOADPDS, and UPDPDS programs which are described earlier in this chapter.

Edit Tables

Edit tables are used by several update and edit programs to define edit checks that are to be performed. The formats of these tables vary, and are given with the description of the update programs in the publication Highway Information System Release 4.Ø - Programming Details. The tables are maintained by the LISTPDS, LOADPDS, and UPDPDS programs described earlier in this chapter.

The Accident-by-Sections Table

The accident-by-sections table contains data needed by the accident-by-sections software. The format of this table is given with the description of the accident-by-sections software in the publication Highway Information System Release 4.Ø - Programming Details. The table is maintained by the LISTPDS, LOADPDS, and UPDPDS programs described earlier in this chapter.

HIS.TABLES Member Names

PROGNAM	Program name table
PASSPARM	Passed parameter table
PARM	Instruction parameter table
EQUIV	Equivalence table
CITIES	City table
COUNTY	County table
PROJECT	Project classification table
SURFTYP	Surface type table
SUFF	Sufficiency table
PTW	PTW table
LOADMOD	Load module table
BDG	File table - bridge file
DAC	File table - accident detail file
RLG	File table - roadlog file
RRX	File table - railroad file
SKD	File table - skid file
SUF	File table - sufficiency file
TRF	File table - traffic file
TRM	File table - true mileage file
USA	File table - urban sign inventory file - assembly records
USS	File table - urban sign inventory file - sign records
USR	File table - urban sign inventory file - remark records
VAC	File table - accident vehicle file
ACDSELCT	Select table - accident directory file
BDGSELCT	Select table - bridge file
DACSELCT	Select table - accident detail file
RLGSELCT	Select table - roadlog file
RRXSELCT	Select table - railroad file
SKDSELCT	Select table - skid file
SUFSELCT	Select table - sufficiency file
TRFSELCT	Select table - traffic file
USNSELCT	Select table - urban sign inventory file
VACSELCT	Select table - accident vehicle file
ACCEDIT1	Edit table - accident
SKDED1	Edit table - skid
SKDED2	Edit table - skid
RAMPLOCN	Edit table - skid
USNEDxxx	Edit table - urban sign inventory (xxx is city number)
ACDLST	Field-list table - LIST-FA-ACC-DIREC
BDGLIST	Field-list table - Bridge LIST
SKDLSTØØ	Field-list table - Skid LIST
SKDLSTØ1	Field-list table - Skid LIST
ACCSECT	Accident-by-sections table

CHAPTER 12

MISCELLANEOUS FILES

The Grid Table

The record format of the grid table is:

Columns	Length	Format	Contents
1-2	2	fixed(3)	City number
3-5	3	fixed(4)	X-coordinate
6-8	3	fixed(4)	Y-coordinate
9-48	4Ø	char(4Ø)	Intersection name
49-52	4	char(4)	Route number of first intersecting street
53-56	4	char(4)	Route number of second intersecting street
57-6Ø	4	char(4)	Route number of third intersecting street
61-8Ø	2Ø		Unused

The Defense Cross-Reference File

The record format of the defense cross-reference file is:

Columns	Length	Format	Contents
1	1	char(1)	Federal aid route system
2-6	5	char(5)	Federal aid route number
7-15	9	char(9)	Starting milepoint of defense section
16-24	9	char(9)	Ending milepoint of defense section
25-29	5	char(5)	Defense section number
3Ø	1	char(1)	Direction indicator
31-38	8	char(8)	Inventory route number
39-43	5	char(5)	Latitude
44-49	6	char(6)	Longitude
5Ø-8Ø	31		Unused

The DEFRDI Subroutine - DEFRDI is a file access subroutine for reading the defense cross-reference file sequentially. It contains entry points DEFRDI and DEFRDIC. To read a record, call DEFRDI and pass a pointer. Upon return, the pointer contains the address of a record. At end-of-file, a null pointer is returned. Upon completion, call DEFRDIC (no arguments are passed) to close the file.

The DEFPNT Subroutine - DEFPNT determines whether a milepoint is located within a defense section. Pass the char(15) milepoint and a binary fixed(15) return code. Return codes are:

- Ø Milepoint not within defense section
- 4 Milepoint within defense section

The Maintenance Division Cross-Reference File

The record format of the maintenance division cross-reference file is:

Columns	Length	Format	Contents
1-2	2	char(2)	Maintenance division
3-17	15	char(15)	Starting key
18-32	15	char(15)	Ending key
33-43	11		Unused

The Maintenance Section Cross-Reference File

The record format of the maintenance section cross-reference file is:

Columns	Length	Format	Contents
1-2	2	char(2)	Maintenance division
3-6	4	char(4)	Maintenance section
7-21	15	char(15)	Starting key
22-36	15	char(15)	Ending key
37-8Ø	44		Unused

The City Route Cross-Reference Files

The record format of the city route cross-reference files is:

Columns	Length	Format	Contents
1	1	char(1)	Delete byte
2-4	3	char(3)	City number
5-8	4	char(4)	Route number
9-11	3	char(3)	Bearing - A direction
12-14	3	char(3)	Bearing - B direction
15-39	25	char(25)	Street name

The Sign Code Cross-Reference File

The record formats of the two sign code cross-reference files are identical. The two files differ only in the location of the key - one is keyed on federal code number and the other is keyed on Montana code number. The order of records in the file may be different in the two files. The record format is:

Columns	Length	Format	Contents
1	1	char(1)	Delete byte
2	1	char(1)	Non-standard Montana code indicator
3-12	1Ø	char(1Ø)	Montana code number
13-14	2	char(2)	Supplemental code
14-21	7	char(7)	Federal code number
22-23	2	char(2)	Sequence number
24	1	char(1)	Non-standard federal code indicator
25-27	3	char(3)	Horizontal dimension
28-3Ø	3	char(3)	Vertical dimension
31-32	2	char(2)	Color
33	1	char(1)	Shape
34-8Ø	47	char(47)	Description

CHAPTER 13

MISCELLANEOUS SUBROUTINES

The PRINT Subroutine

PRINT should be used by all programs written to execute under HIS for producing printed output. It automatically implements all of the HIS print options documented in the publication Highway Information System Release 4.Ø - User's Manual.

All mainline programs that call PRINT or any of the other HIS subroutines must include the following declarations:

```
DECLARE PRINTER POINTER EXTERNAL,
        name1 CHAR(132),
        name2 CHAR(132) BASED(PRINTER);

PRINTER = ADDR(name1);
```

All subroutines that call PRINT must include the following declarations:

```
DECLARE PRINTER POINTER EXTERNAL,
        name2 CHAR(132) BASED(PRINTER);
```

Suggested names are PBUF for name1 and PRNTR for name2. If the declaration for name1 includes STATIC, the char(132) area is easier to find in a dump.

PRINT Entry Point - Call PRINT to print a line. The line to be printed is first placed into PRNTR. Pass a fixed(1) value to indicate spacing control:

Ø Suppress spacing
1 Single space
2 Double space
3 Triple space
4 Space 4 lines
5 Space 5 lines
6 Space 6 lines
7 Space 7 lines
8 Space 8 lines
9 Advance to new page

Spacing is performed before the line in PRNTR is printed.

PRINTA Entry Point - Call PRINTA instead of PRINT when it is desired to print a line on the current page if there is room for a specified number of lines on that page and to advance to a new page otherwise. Pass a fixed(1) value that specifies spacing control (see PRINT entry point) and a fixed(3) value that specifies the number of lines that must remain on the page after printing if the line is to be printed on the current page.

PRINTB Entry Point - Call PRINTB instead of PRINT when it is desired to print a line only if it fits on the current page. Pass a fixed(1) value that specifies spacing control (see PRINT entry point).

SETPOS Entry Point - Call SETPOS to space one or more lines without printing a line. Pass a fixed(1) value that specifies spacing control (see PRINT entry point).

SETPOSA Entry Point - Call SETPOSA instead of SETPOS if it is desired to jump to a new page if less than a specified number of lines remain on the current page. Pass a fixed(1) spacing control value (see PRINT entry point) and a fixed(3) value that specifies the number of lines that must remain on the current page after spacing.

SETNEW Entry Point - A call to SETNEW indicates that a new page must be started when PRINT, PRINTA, SETPOS, or SETPOSA is subsequently called. The printer is not advanced to a new page until a subsequent call to one of these entry points to allow page headings to be changed between the SETNEW call and the call to one of these entry points.

SETHDG Entry Point - Call SETHDG to establish page headings. Pass a binary fixed(15) value and a char(132) variable. Specify the number of headings (Ø-2Ø) in the binary fixed variable. Pass the first or only heading line in the char(132) variable. If more than one heading line is specified in the binary variable, the remaining lines must physically follow the first line in char(132) variables (eg., declare the headings in an array or structure in char(132) elements). The subroutine saves the number and address of

of the headings rather than the headings themselves to save core and to allow the calling program to modify the headings without performing another SETHDG call. Any time the printer is advanced to a new page, the heading lines are printed. The SETHDGS entry point can be called instead of SETHDG by PL/I optimizing programs only. Declaration and calling sequence is the same as SETHDG except that OPTIONS (ASM INTER) must be included in the entry point declaration.

SETINST Entry Point - SETINST returns a pointer that contains the address of the HIS instruction. Pass a pointer variable.

SETLINK Entry Point - SETLINK returns a pointer that contains the address of the print control block. Pass a pointer variable.

SETDD Entry Point - SETDD closes the print file and re-opens it using a program-specified ddname. Pass a char(8) ddname.

DUMP Entry Point - DUMP closes the print files and issues a user-1ØØ abend. The print files are closed so that the print buffers are not lost. No arguments need to be passed, but arguments that are passed can be easily located in the dump. Follow these steps:

1. Near the beginning of the dump, find the load list (labeled as LOAD LIST).
2. Locate the LPRB entry that has the format LPRB xxxxx NM HIS2ØØØØ. Note the address listed after LCT/EP xxxxxxxx (the last six digits is the entry point of the supervisor).
3. Locate the supervisor entry point in the dump using the address in LCT/EP.
4. Near this entry point (within a couple of pages) will be found the character string "PRINT STORAGE". The contents of general register 1 (used to pass arguments) is located in the fullword immediately following this character string. The value located is the contents of register 1 when DUMP was entered.

Sample Programs - Use of PRINT - These programs read and print 8Ø-byte records from a file whose ddname is specified in the HIS command. The first program is in PL/I and the second is in assembler.

```
/* SAMPLE PROGRAM - USE OF PRINT SUBROUTINE */
LISTIT:  PROCEDURE OPTIONS (MAIN);   /* MAINLINE PROGRAM */
/* PRINT SUBROUTINE AND HIS INSTRUCTION */
DECLARE PRINT   ENTRY (FIXED(1)),
        PRINTA  ENTRY (FIXED(1),FIXED(3)),
        SETHDG  ENTRY (BIN FIXED,CHAR(132)),
        SETINST ENTRY (POINTER),
        DUMP    ENTRY,
        PTRINS  POINTER,  /* FOR INSTRUCTION */
        1  INS  BASED(PTRINS),
                2  FILLER CHAR(118),
                2  DDNAME CHAR(8),
        PRINTER POINTER EXTERNAL,
        PBUF    CHAR(132) STATIC,
        PRNTR   CHAR(132) BASED(PRINTER),
        HEADING(3) CHAR(132) STATIC INIT
                ('CARD LIST PROGRAM',' ',' ');
/* INPUT FILE */
DECLARE PTRCARD POINTER,  /* FOR INPUT RECORDS */
        CARD    CHAR(8Ø) BASED(PTRCARD),
        INFILE  FILE INTERNAL RECORD;
ON ERROR BEGIN;  /* ERROR HANDLER */
   CLOSE FILE (SYSPRINT);  /* SAVE PL/I ERRMSG BUFFERS */
   CALL DUMP;  /* ISSUE USER-1ØØ ABEND */
   END;
PRINTER = ADDR(PBUF);
CALL SETINST (PTRINS);
CALL SETHDG (3,HEADING(1));
OPEN FILE(INFILE) TITLE(INS.DDNAME);
ON ENDFILE(INFILE) GOTO DONE;
READ FILE(INFILE) SET(PTRCARD);  /* FIRST REC IS HEADING LINE */
HEADING(3) = CARD;
LOOP:   READ FILE (INFILE) SET (PTRCARD);
        PRNTR = CARD;
        CALL PRINT (1);
        GOTO LOOP;
DONE:   CLOSE FILE(INFILE);
        END LISTIT;
```

```
*---- SAMPLE PROGRAM - USE OF PRINT SUBROUTINE ---*
LISTIT   START
RS1      EQU   2   Scratch register
RDCB     EQU   3   For DCB address
RPRNT    EQU   4   For PRINTER address
RINS     EQU   5   For instruction address
RBASE    EQU   6   Program base register
         EXTRN PRINTER
         USING *,RBASE
         USING INS,RINS
         USING IHADCB,RDCB
         USING PRINTER,RPRNT
         STM   14,12,12(13)      Save registers
         LR    RBASE,15          Load program base register
         ST    13,SAVEAREA+4     Set up savearea linkage
         LR    RS1,13
         LA    13,SAVEAREA
         ST    13,8(RS1)
         L     RS1,=V(PRINTER)   Load PRINTER address
         L     RPRNT,Ø(RS1)      Load PRNTR address
         MVI   PRINTER,C' '      Fill PRNTR with blanks
         MVC   PRINTER+1(131),PRINTER
         CALL  SETINST,(PTRINS)   Retrieve instruction
         L     RINS,PTRINS
         CALL  SETHDGS,(#HDG,HEADING)   Establish headings
         LA    RDCB,INFILE   Store ddname in DCB
         MVC   DCBDDNAM,INSDDNAM
         OPEN  INFILE        Open the DCB
         GET   INFILE        Read first record
         MVC   HEADING3(8Ø),Ø(1)   Store in heading
LOOP     GET   INFILE              Read a card
         MVC   PRINTER(8Ø),Ø(1)    Move into PRNTR
         CALL  PRINT,(SPACE1)      Print the card
         B     LOOP                Branch to LOOP
DONE     CLOSE INFILE      Close the file
         FREEPOOL INFILE   Free the buffer pool
         L     13,SAVEAREA+4   Restore register 13
         RETURN (14,12),RC=Ø   Restore registers and return
SAVEAREA DS    18F     Register save area
PTRINS   DS    A       Pointer for SETINST call
#HDG     DC    H'3'    Number of headings for SETHDGS call
HEADING  DC    CL132'CARD LIST PROGRAM'
         DC    CL132' '
HEADING3 DC    CL132' '
         LTORG
INFILE   DCB   DSORG=PS,MACRF=GL,EODAD=DONE
```

```
INS      DSECT          Dummy section for instruction format
         DS    CL118
INSDDNAM DS    CL8

         DCBD           Generates dummy section with DCB field names

         END
```

The CHECKDD Subroutine

CHECKDD checks the IBM system control blocks to determine whether a DD statement has been included with a run. Pass the char(8) ddname and a pointer variable. If the DD statement has been included, the address of the TIOT entry for the ddname is returned in the pointer. If the DD statement has not been included, a null pointer is returned.

The DUMPDD Subroutine

DUMPDD checks the IBM system control blocks to determine whether a DD statement has been included with a run. Pass the char(8) ddname and two binary fixed(15) variables. Set the first binary variable to one of these values before calling DUMPDD:

- Ø   Issue user-1ØØ abend if DD statement has not been included
- 1   Return control whether or not DD statement has been included

The return code will be set to one of these values by DUMPDD:

- Ø   DD statement has been included
- 4   DD statement has not been included

If the DD statement has not been included, DUMPDD prints a message in the following format before returning or issuing an abend:

```
***** name DD STATEMENT MISSING *****
```

The GETDAY Subroutine

GETDAY returns the day of the week of a passed date. It can be called only by PL/I optimizing programs. The date passed must be a 2Øth century date. Pass three fixed(2) variables and two fixed(3) variables. Pass the date in the first three variables (month, day, then year). The julian day is returned in the first

fixed(3) variable. The day of the week is returned in the second fixed(3) variable. Codes used for day of week are:

1 Saturday
2 Sunday
3 Monday
4 Tuesday
5 Wednesday
6 Thursday
7 Friday

The GETDATE Subroutine

GETDATE returns the current date. Pass three fixed(2) variables. The date is returned as month, day, then year.

The DATEDIT Subroutine

DATEDIT tests for a valid 2Øth century date. Pass three fixed(2) variables and a binary fixed(15) variable. Pass the date in the fixed(2) variables (month, day, then year). The binary fixed(15) variable is a return code:

Ø Valid date
1 No date (each of the fixed(2) variables contains zero)
2 Date is in error

The DATE1 Subroutine

DATE1 converts a date to mm/dd/yy format for printing. Pass three fixed(2) variables and a char(8) variable. Pass the date in the fixed(2) variables (month, day, then year). The date is returned in mm/dd/yy format in the char(8) variable.

The DATE2 Subroutine

DATE2 converts a date to mmm dd,yyyy (eg., OCT 22,1975) format for printing. Pass three fixed(2) variables and a char(11) variable. Pass the date in the fixed(2) variables (month, day, then year). The date is returned in mmm dd,yyyy format in the char(11) variable.

The DATE3 Subroutine

DATE3 returns the current date in mm/dd/yy format. Pass a char(8) variable.

The DATE4 Subroutine

DATE4 returns the current date in mmm dd,yyyy format (eg., OCT 22,1975). Pass a char(11) variable.

The CVTLOCN Subroutine

CVTLOCN analyzes the contents of the LOCATION/CITY/COUNTY parameters of HIS commands. Only one field is alloted in the instruction for these parameters. If more than one was coded, the one which was physically last on the command is the one used. Pass the char(18) location/city/county field from the instruction, a fixed(3) variable, a fixed(2) variable, and a binary fixed(15) variable. The binary fixed(15) variable is used as a return code:

Ø	LOCATION/CITY/COUNTY parameter in error
1	No location/city/county present (all blanks passed)
2	LOCATION=STATEWIDE
3	CITY=name
4	COUNTY=name
5	LOCATION=EVERYTHING

If the return code is 3, the city number is returned in the fixed(3) variable. If the return code is 2, the registration county number is returned in the fixed(2) variable.

The CNVSTAT Subroutine

CNVSTAT converts a char(2) state code to a state number and vice versa, and supplies the state name. Pass a char(2) variable, a fixed(2) variable, and a char(2Ø) variable. The state numbers are assigned internally by CNVSTAT and can be used as array bounds. To convert a state code to a number, pass the code in the char(2) variable (if the code is unknown, a number of 66 and a name of *******ERROR******* are returned). To convert a number back to its state code, pass blanks in the char(2) variable and the number in the fixed(2) variable. If an unknown number is passed, a code of ** and a name of *******ERROR******* are returned.

The BANNER Subroutine

BANNER prints a banner sheet for a passed character string. Declare the pointer LOCAT as external and set it to the address of a char(13) string. The first 12 characters are the characters to be printed. The last character is the character to be used in the printing. Each character is printed in an 8-character by 8-character square. If the last character is blank, each character is printed using its own character. An example program is:

```
/* SAMPLE PROGRAM - USE OF BANNER SUBROUTINE */
SAMPLE:  PROCEDURE OPTIONS (MAIN);
DECLARE PRINT   ENTRY (FIXED(1)),
        SETPOS  ENTRY (FIXED(1)),
        BANNER  ENTRY,
        LOCAT   POINTER EXTERNAL,
        1  STR,
           2  STRING CHAR(12),
           2  CHAR   CHAR(1);

        LOCAT = ADDR(STR);
        STR.STRING = 'III';
        STR.CHAR   = '*';
        CALL BANNER;
        CALL SETPOS (3);
        STR.CHAR = ' ';
        CALL BANNER;

        END SAMPLE;
```

Note: BANNER always prints 8 lines and does no printer spacing - if spacing is necessary, call SETPOS or SETPOSA of PRINT.

BANNER will print the following for this example:

```
********  ********  ********
********  ********  ********
   **        **        **
   **        **        **
   **        **        **
   **        **        **
********  ********  ********
********  ********  ********

IIIIIIII  IIIIIIII  IIIIIIII
IIIIIIII  IIIIIIII  IIIIIIII
   II        II        II
   II        II        II
   II        II        II
   II        II        II
IIIIIIII  IIIIIIII  IIIIIIII
IIIIIIII  IIIIIIII  IIIIIIII
```

The GETJFCB Subroutine

GETJFCB reads a Job File Control Block (JFCB) from the IBM system control blocks. Pass a char(8) ddname, a char(176) area for the JFCB, and a binary fixed(15) variable for return codes:

Ø	Successful
1	Unsuccessful (DD statement is probably missing)
2	Function disabled

The format of a JFCB is given in the IBM publication IBM System/360 Operating System: System Control Blocks, order number GC28-6628.

A program that processes a JFCB is dependant upon the formats of IBM control blocks and hence may be subject to necessary changes in future versions of the operating system. If the control block formats are changed, GETJFCB will always return a return code of 2. If the calling programs accept a return code of 2 and bypass their JFCB code when this return code is received, they will still operate correctly after such a change.

The GETDSCB Subroutine

GETDSCB reads a Data Set Control Block (DSCB) from IBM system control blocks on disk packs. DSCB's are entries in the Volume Table of Contents (VTOC) located on the disks. The format of various DSCB's is given in the IBM publication IBM System/360 Operating System: System Control Blocks, order number GC28-6628.

When calling GETDSCB, pass a binary fixed(15) option code, four pointer variables, and a binary fixed(15) return code. The option code indicates the type of access provided by the calling program:

Ø	A char(8) ddname is provided
1	A char(5Ø) volume serial number and data set name are provided
2	A char(176) JFCB is provided

The first pointer must contain the address of either a ddname, a volume serial number and data set name, or a JFCB. The DSCB's are returned in the remaining three pointers in this order:

1	Format-1 DSCB - always present if request is valid
2	Format-2 DSCB - present only for indexed-sequential files
3	Format-3 DSCB - present if data set has been extended 4 or more times

These three pointers must contain the address of 14∅-character areas upon entry to GETDSCB. The subroutine places the DSCBs in these areas. 14∅ blanks are placed in any areas which correspond to a DSCB format that did not exist.

Return codes are:

∅	Successful - DSORG unknown
1	Successful - Physical-sequential organization
2	Successful - Indexed-sequential organization
3	Successful - Partitioned organization
4	Successful - Direct organization
5	Unsuccessful - Volume not mounted (RC=4 from OBTAIN)
6	Unsuccessful - Format-1 DSCB not found (RC=8 from OBTAIN)
7	Unsuccessful - I/O error (RC=12 from OBTAIN)
8	Unsuccessful - Invalid workarea pointer (RC=16 from OBTAIN)
9	Unsuccessful - Non-zero return code from GETJFCB subroutine
1∅	Unsuccessful - Invalid option code passed
11	Function disabled

A program that processes a DSCB is dependant upon the formats of IBM control blocks and may need alterations in future versions of the operating system. Should this occur, GETDSCB will always return a return code of 11. Calling programs should accept this return code and bypass DSCB checking when this code is received. If this is done, the programs will continue to operate under future operating systems.

The BACKUP Subroutine

BACKUP produces a backup copy of an indexed-sequential file. Pass two char(8) ddnames - the first the ddname for the input file (indexed-sequential) and the second the ddname for the output file (physical-sequential). DCB information must be available in the data set labels or on the DD statements. The record length specified for the two files must be equal. If not, the message

```
***** RECORD LENGTH OF BACKUP FILE DIFFERS FROM RECORD LENGTH OF
      FILE BEING COPIED *****
```

is printed and a user-1∅∅ abend is issued.

The LOAD Subroutine

LOAD loads an indexed-sequential file from a physical-sequential file. Pass two char(8) ddnames - the first for the output indexed-sequential file and the second for the input physical-sequential file. The record length specified for each of these files in the DD statement or data set labels must be identical. If not, the message

```
***** RECORD LENGTH OF BACKUP FILE DIFFERS FROM RECORD LENGTH OF
      OF FILE BEING CREATED *****
```

is printed and a user-1ØØ abend is issued.

The FETCH Subroutine

FETCH retrieves a load module for execution. A maximum of 2Ø modules can be loaded at any one time. FETCH can only be called by programs written in assembler.

FETCH Entry Point - Call FETCH to retrieve a load module. Pass the address of an 8-byte area that contains the load module name in register 1. The entry point address of the load module is returned in register 15. The following system abends may occur when FETCH is called:

8Ø6 Load module not found
7Ø6 Load module marked not executable by link editor

FETCHD Entry Point - Call FETCHD to delete a load module. Pass the address of an 8-byte area that contains the load module name in register 1. Load modules that are not deleted will be deleted by the supervisor upon return. It is suggested that FETCHD not be called unless the limit of 2Ø modules is reached so that the modules will be in core if a dump is requested by means of a DUMP parameter in the PARM parameter of the EXEC statement.

FETCHX Entry Point - FETCHX is available only to the supervisor. When called, all modules that have been loaded by FETCH are deleted.

The GETTIOT Subroutine

GETTIOT returns the address of the Task Input/Output Table (TIOT). This IBM control block can be used to determine whether a DD statement has been included with a run. The format of the TIOT is documented in the IBM publication IBM System/360 Operating System - System Control Blocks, order number GC28-6628. When calling GETTIOT, pass a pointer variable. The address of the TIOT is returned in the pointer.

The SELTEST Subroutine

SELTEST includes all of the software of the select subsystem. It is used by file access subroutines that incorporate select statements. Only one subroutine can call SELTEST in any one run.

SELTESTI Entry Point - Call SELTESTI to read the select statement and to initialize any necessary files. SELTESTI must not be called unless a select statement has been included on the command. Pass the char(3) file name. A list of file names is provided with the description of the select tables in chapter 11. For the accident files, pass the name ACC instead of ACD, DAC, or VAC.

SELTEST Entry Point - Call SELTEST to check a record for inclusion. Pass a pointer that contains the record address (the record must be a record from the file specified in the SELTESTI call). For the accident files, pass a pointer that contains the address of the following control block:

Columns	Length	Format	Contents
1-4	4	Pointer	Address of directory record or null
5-8	4	Pointer	Address of detail record or null
9-12	4	Pointer	Address of vehicle array or null
13-14	2	Binary	Number of vehicle records

(At least one pointer must be non-null)

Upon return, register 15 contains a return code:

Ø Accept record
1 Skip record

SELTESTC Entry Point - Upon completion, call SELTESTC to close files and release core. No arguments are passed.